The Dissertation Committee for Selin Erzeybek Balan Certifies that this is the approved version of the following dissertation:

Characterization and Modeling of Paleokarst Reservoirs Using Multiple-Point Statistics on a Non-Gridded Basis

Committee:

Sanjay Srinivasan Supervisor

Steven L. Bryant

Xavier Janson

Larry W. Lake

Christopher K. Zahm

Characterization and Modeling of Paleokarst Reservoirs Using Multiple-Point Statistics on a Non-Gridded Basis

by

Selin Erzeybek Balan, B.S.; M.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

December 2012

Dedication

To my beloved husband Huseyin Onur; for his ever-lasting love, continuous support and courage. He makes my life full of joy, peace and happiness.

To my dear parents, especially to my mother Nimet and my brother Yunus Serhat; for their valuable love and support.

To Mustafa Kemal ATATURK, the founder of the Republic of Turkey; for giving me hope and courage. I will not be able to accomplish any of my goals without his presence, his principles and his dedication to my country.

Acknowledgements

I would like to express my deepest gratitude to my supervisor Dr. Sanjay Srinivasan for his guidance and support during my Ph.D. education. He has been a great mentor and I am grateful for his contributions to my academic career. Dr. Srinivasan has a unique way of teaching Geostatistics and I had the most enlightening learning experience under his guidance. I feel very fortunate to be a part of his research team during my four years at UT-Austin. He introduced me the great area of Geostatistics and I am thankful to him for sharing his great knowledge and his expertise with me. He has been very patient and encouraging during my ups and downs in my challenging Ph.D. journey. Dr. Srinivasan, thank you for being a great supervisor.

I am also thankful to my committee members for their valuable contributions and suggestions for my dissertation. I would like to acknowledge Dr. Xavier Janson for his guidance, advices and contributions throughout my study. It was a great opportunity for me to have his suggestions while evolving my dissertation. I am also thankful to Dr. Larry Lake and Dr. Chris Zahm for their evaluating my progress. Their comments and were very helpful and enhancing. I also thank Dr. Steven Bryant for his contributions both in my research and in my academic training. Learning Transport Phenomena from his perspective has been a valuable experience that allows me to deeply understand reservoir processes and to be a better researcher.

I would like to thank my dear friends; Dr. Cigdem Omurlu Metin, Tolga Metin, Dr. Cesar Mantilla, Sayantan Bhowmik, Prince Azom, Dr. Juliana Leung, Ankesh Anupam and Brandon Henke. Special thanks to my group members; Harpreet, John, Hoonyoung, Kwangjin, Travis, Arti, D.J. and many others for sharing their time and

friendship with me. I want to acknowledge Jin Lee, for her friendship and support in administration issues; Frankie Hart, for her guidance, and Roger Terzian, for his technical support whenever I needed.

This study will not be completed without the generous financial support from Saudi Aramco. Also, I acknowledge Kinder Morgan Energy Company for providing field data and especially Jesse Garnett White for taking his time to send me the core images. I am also thankful to International Association for Mathematical Geosciences, Graduate School and Graduate Engineering Council for awarding me travel grants. I want to thank Schlumberger for providing Petrel, Computer Modeling Group for proving IMEX, and GsTL programmers.

Finally but not lastly, I am thankful to my family; my mother Nimet, my father Selim, my dear brother Yunus Serhat and my wonderful husband Huseyin Onur. Without their support, I will not be able to reach my goals. I would like to especially thank to my husband Onur, he is the most supportive and the most loving husband ever. We shared this challenge together, spent our most memorable engagement days by studying the qualifying exams and rehearsed our proposal exam presentations instead of rehearsing our wedding vows. We shared this challenge together and we will share rest of our lives together. His support helped me to complete my accomplishments.

Characterization and Modeling of Paleokarst Reservoirs Using Multiple-Point Statistics on a Non-Gridded Basis

Selin Erzeybek Balan, Ph.D.

The University of Texas at Austin, 2012

Supervisor: Sanjay Srinivasan

Paleokarst reservoirs consist of complex cave networks, which are formed by various mechanisms and associated collapsed cave facies. Traditionally, cave structures are defined using variogram-based methods in flow models and this description does not precisely represent the reservoir geology. Algorithms based on multiple-point statistics (MPS) are widely used in modeling complex geologic structures. Statistics required for these algorithms are inferred from gridded training images. However, structures like modern cave networks are represented by point data sets. Thus, it is not practical to apply rigid and gridded templates and training images for the simulation of such features. Therefore, a quantitative algorithm to characterize and model paleokarst reservoirs based on physical and geological attributes is needed.

In this study, a unique non-gridded MPS analysis and pattern simulation algorithms are developed to infer statistics from modern cave networks and simulate distribution of cave structures in paleokarst reservoirs. Non-gridded MPS technique is practical by eliminating use of grids and gridding procedure, which is challenging to apply on cave network due to its complex structure. Statistics are calculated using commonly available cave networks, which are only represented by central line coordinates sampled along the accessible cave passages. Once the statistics are calibrated,

a cave network is simulated by using a pattern simulation algorithm in which the simulation is conditioned to sparse data in the form of locations with cave facies or coordinates of cave structures. To get an accurate model for the spatial extent of the cave facies, an algorithm is also developed to simulate cave zone thickness while simulating the network.

The proposed techniques are first implemented to represent connectivity statistics for synthetic data sets, which are used as point-set training images and are analogous to the data typically available for a cave network. Once the applicability of the algorithms is verified, non-gridded MPS analysis and pattern simulation are conducted for the Wind Cave located in South Dakota. The developed algorithms successfully characterize and model cave networks that can only be described by point sets. Subsequently, a cave network system is simulated for the Yates Field in West Texas which is a paleokarst reservoir. Well locations with cave facies and identified cave zone thickness values are used for conditioning the pattern simulation that utilizes the MP-histograms calibrated for Wind Cave. Then, the simulated cave network is implemented into flow simulation models to understand the effects of cave structures on fluid flow. Calibration of flow model against the primary production data is attempted to demonstrate that the pattern simulation algorithm yields detailed description of spatial distribution of cave facies. Moreover, impact of accurately representing network connectivity on flow responses is explored by a water injection case. Fluid flow responses are compared for models with cave networks that are constructed by non-gridded MPS and a traditional modeling workflow using sequential indicator simulation. Applications on the Yates Field show that the cave network and corresponding cave facies are successfully modeled by using the non-gridded MPS. Detailed description of cave facies in the reservoir yields accurate flow simulation results and better future predictions.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Overview

Paleokarst reservoirs consist of complex geological features such as collapsed cave facies, substratal deformation structures, fracture networks associated with dissolution and collapse of cave structures. Thus, it is challenging to characterize these complex heterogeneity features and understand their corresponding effects on fluid flow. Although significant hydrocarbon resources are in paleokarst settings (e.g. the Ellenburger Group carbonate reservoirs, Ordos and Tarim Basin reservoirs in China, Yates Field of West Texas and Super-K zones of Ghawar Field), detailed reservoir characterization and methods for realistically modeling cave features are limited (Dogru et al., 2001; Massonat and Pernarcic, 2002; Botton-Dumay et al., 2002). On the other hand, most of the studies in paleokarst reservoirs are restricted to structural, stratigraphic and sedimentological analysis (Loucks, 1999; Loucks and Mescher, 2001). These studies yield valuable information on paleokarst formation mechanisms and associated facies. There are recent attempts to model karst networks and understanding spatial distribution of passages (Henrion et al., 2008; Borgh et al., 2012).

Due to significant uncertainty associated with subsurface conditions prevalent at the time of paleocave formation and their subsequent collapse, it is necessary to incorporate details of the reservoir formation processes within a stochastic modeling framework in order to obtain realistic description of reservoir heterogeneity while at the same time represent the residual uncertainty. Stochastic techniques involving two-point and multiple-point statistics are practical tools to model complex geologic features and some of them might be applicable to characterize karst networks. Most common geostatistical methods such as Kriging and Sequential Gaussian Simulation (SGSIM), involve two-point statistics. These methods are variogram-based and require that the

modeled reservoir attribute exhibit Gaussianity. Although algorithms like co-kriging enables the integration of different data groups, the multivariate Gaussian assumption restricts their applicability (Strebelle, 2002). The limitation of Gaussianity can be overcome in indicator-based techniques (Goovaerts, 1997) such as sequential indicator simulation (SISIM). However, it is difficult to reproduce complex geological features such as faults, fractures and fluvial channels by constraining the models to two-point (variogram) statistics (Strebelle, 2002). On the other hand, recently developed MPS algorithms are capable of representing complex geologic structures as well as integrating data from different sources (Caers et al., 2000; Strebelle, 2002; Caers, 2002; Caers and Zhang, 2004; Eskandaridalvand, 2008). In these MPS algorithms, pattern statistics are inferred using spatial templates and realistic analog/training images. Complex geological features such as fracture networks can be represented accurately using MPS (Liu and Srinivasan 2005).

1.2 PREVIOUS APPROACHES TO MODEL KARST NETWORKS

Since several mechanisms control the formation of karst networks, it is difficult to describe spatial distribution of the cave structures and corresponding effects on fluid flow by using only geological information from analogs. Moreover, the distribution of karstic structures is controlled by presence of fractures and bedding planes. Thus, stochastic simulation algorithms honoring structural geology have been recently developed for karst network modeling (Henrion et al., 2008; Borghi et al., 2012).

Henrion and others (2008) proposed a new methodology to generate stochastic models of 3D cave systems by combining object-based and variogram-based geostatistical algorithms. The proposed algorithm simulates 3D cave networks using topology and geometry of karst conduits. Initially, they generate discrete fracture networks on a matrix grid by using object-based simulation. Then, rock matrix and

discontinuous medium are discretized into a graph of connectivities. Conditioning data such as field measurements, water level, source and sink information are integrated on the graph. Consequently, preferential flow pathways are extracted using a graph search algorithm to obtain spatial distribution of the cave system (Henrion et al., 2008). Finally, geometry of the cave conduits is simulated by using grid distance to the flow pathway maps that are constructed by Sequential Gaussian Simulation (Henrion et al., 2008) (Figure 1.1). While this methodology does not simulate the dissolution process, Henrion et al. (2008) concluded that geostatistical perturbation of the distances to preferential flow paths could yield realistic models of complex cave systems. Despite its potential, the algorithm has drawbacks; as noted by the authors, it is based on simulation of fracture networks and the simulation of multiple fracture networks is crucial (Henrion et al., 2008). Moreover, extent of cave passages, i.e. Euclidean distance map, is based on distance cutoffs. Thus, the variogram used in Sequential Gaussian Simulation should be defined carefully and it is essential to inherit the overall geology of cave system. Furthermore, this study was not advanced enough to address the effect of cave network on fluid flow.

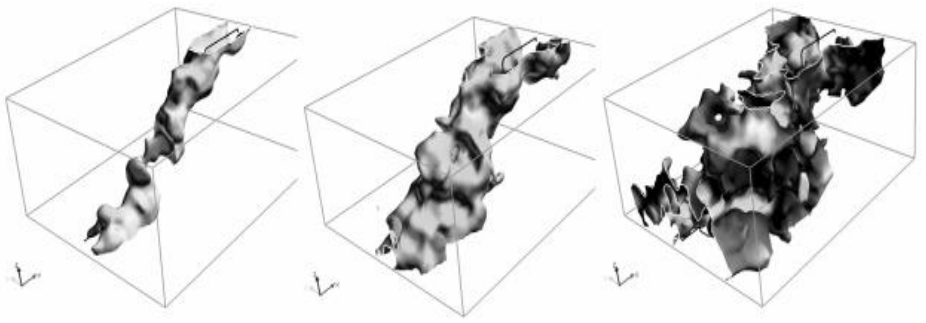


Figure 1.1- Karst Network Realizations (Henrion et al., 2008)

More recently, Borghi et al. (2012) presented a methodology for stochastic simulation of 3D karstic conduits of epigenic systems. This algorithm integrates knowledge about cave formation processes and field measurements of aquifers and springs. The methodology consists of four main steps; (1) construction of a 3D geological model of the region, (2) stochastic simulation of heterogeneity features such as fractures, bedding planes and inception horizons, (3) identification of potential inlets and outlets of the systems, base level and different phases of karstification, (4) generating karst network using a fast marching algorithm. The technique provides realizations of 3D karst reservoir models that are conditioned to the regional geology, local heterogeneities and regional flow conditions. First, regional scale structures that control the formation of karstic networks are modeled. Sedimentary and hydro-stratigraphic units are defined accurately and the authors used **Geomodeller3D** software which handles complex geological structures for such purposes. Second step is the construction of fracture network using a Boolean object model on a regular rectangular grid. Inception horizons represented by cells with increased permeability or iso-thickness surfaces representing relative thickness of the karstic formation are modeled. The next step is determination of inlets and outlets of the karst aquifer system. Inlets of the system such as dolines and sinkholes are identified by hydrogeological analysis and tracer tests. Outlets such as springs and spring zones are also determined by hydrogeological analysis. The final step of the algorithm is generation of karstic networks using a Fast-Marching Algorithm whereby the conduits propagate from springs, i.e. known aquifer outlets (Borghi et al., 2012) (Figure 1.2). The propagation is controlled by a velocity field which is defined by combining different hydrogeological features such as fractures, bedding planes, aquitard formations and inception horizons.

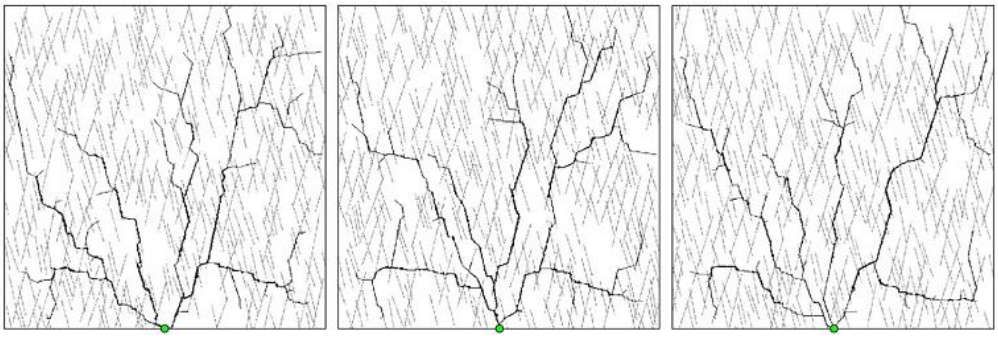


Figure 1.2- 2D Map of Karst Network Realizations (Borghi et al., 2012) (Inlets: Starting points of the conduits; Outlet: Green point)

The methodology presented by Borghi et al. (2012) enables one to integrate different sets of information. As stated by the authors, realizations are based on stochastic models of inlet locations, fracture network and outlet locations. Thus, it is essential to have accurate information on structural geology and hydrogeological conditions. Moreover, this study is restricted to epigenic type of karst systems; it does not handle hypogenic or island type of networks which are recognized in Ghawar field of Saudi Arabia and Yates field of West Texas. Furthermore, the algorithm requires a fully described geologic model of the entire system in order to simulate karst network and this procedure might not be practical in cases with limited data.

Some of the important hydrocarbon resources are in karst modified reservoirs such as Ghawar Field in Saudi Arabia and Kirkuk Field in Iraq. On the other hand, most of the studies in paleokarst reservoirs are restricted to structural, stratigraphical and sedimentological aspects (Tinker et al., 1995; Loucks, 1999). Moreover, detailed reservoir characterization and methods for realistically modeling cave features are limited (Henrion et al., 2008). Complex structure of the paleokarst facies are modeled by using high permeability streaks (Dogru et al., 2001), dual-porosity/ dual-permeability formulations (Uba et al., 2007) in reservoir engineering applications. These studies are conducted for history matching purposes and they are usually based on modeling

distribution of paleokarst structures using well test analysis and core information. On the other hand, they do not address the issue of characterization and modeling of paleokarst networks using a solid methodology. Thus, an algorithm is needed to understand spatial distribution of cave networks and to integrate cave facies into flow simulation models for accurate reservoir description and future flow predictions.

1.3 RESEARCH OBJECTIVE

In the literature, we are able to find detailed descriptions of karst formation and evolution (Ford, 1988; Palmer, 1991; Esteban and Wilson, 1993; Labourdette et al., 2007). Also, cave collapse and associated paleokarst facies have been extensively investigated (Loucks, 1999, 2007; Loucks and Mescher, 2001). However, those background geological concepts have not necessarily made their way to reservoir engineering models that can help predict the flow performance of fluids in such reservoirs. Recently, Henrion et al. (2008) and Borghi et al. (2012) proposed methods that enable data conditioning for modeling karst/cave networks but they require a fully described stratigraphic model of the karstic system, a pre-existing fracture network for karst/cave formation and multiple steps for network construction. Moreover, their approaches condense the geological details related to the process of karst formation into a set of heuristic rules (such as threshold for the distance map or rules for marching the random walker) and do not fully exploit the pattern characteristics observed in analog systems.

Our attempt is to develop a new multiple point statistics' based algorithm for simulating karst networks using sparse data. Our focus will be to infer pattern statistics regarding the spatial distribution of cave networks from modern cave analogs and subsequently combine the statistics together with reservoir-specific conditioning information and prior geologic knowledge to simulate the spatial distribution of cave

facies. In reservoir characterization studies, MPS have been widely used in modeling complex geologic structures and fracture networks while integrating different data sets. Thus, curvilinear geological features of paleokarst reservoirs, similar to fracture networks, can be represented accurately using MPS. However, in traditional MPS implementations, a gridded training image is required for inferring the statistics. However, modern caves that are analogs to collapsed cave features are represented by surveys reporting XYZ coordinates of cave central line. Therefore, it is not practical to perform gridding and apply a rigid gridded template to calibrate statistics using such "point" data surveys.

To address these issues, we have developed an MPS algorithm that works on a non-gridded basis to characterize cave and paleokarst networks using "point-set" information. Contrary to the conventional MPS algorithms, non-gridded technique does not require any gridding of the "point-set" data which might be challenging for cave surveys with more than thousands of measurements. The algorithm basically consists of MPS analysis using flexible spatial templates and subsequent pattern simulation using the calibrated statistics. The proposed approach captures spatial distribution of connected cave passages which are curvilinear and complex structures. Moreover, cave networks are successfully simulated while honoring the conditioning information. Furthermore, cave connections are accurately predicted by the spatially flexible templates which are capable of handling complex orientations.

The algorithm is demonstrated on several synthetic data sets first and later applied to simulate fluid flow in the reservoir model for the Yates Field. Fluid flow simulation is performed on the model for the Yates Field in order to assess the importance of accurately depicting the spatial connectivity of karst networks on field development plans and design of recovery processes.

1.4 DISSERTATION OUTLINE

In Chapter 2, literature review of paleokarst reservoirs and stochastic modeling methods is given. Karst formation mechanisms, factors affecting karst formation, general information on paleokarst reservoirs and important hydrocarbons resources in paleokarst formations are presented. Moreover, general overview of stochastic modeling techniques is provided. In Chapter 3, the non-gridded MPS analysis algorithm and its corresponding properties are explained with examples of 2D and 3D synthetic data sets. In Chapter 4, pattern simulation of karst networks and further details of simulation options are discussed. The algorithm is demonstrated by using synthetic data and corresponding results are given. Chapter 5 presents the approach for modeling cave thickness which is performed after the construction of karst network. Application of the developed methodology on a modern cave analog is demonstrated in Chapter 6. Further application of the non-gridded MPS to develop the reservoir model for the Yates Field of West Texas, is presented in Chapter 7. That chapter also presents the results of fluid flow simulations performed using the detailed description of cave networks in the Yates field. The dissertation is concluded in Chapter 8 where the main findings of the research and suggested future work are presented.

Chapter 2: Literature Review

2.1 Geological Background On Paleokarst Reservoirs

This dissertation mainly focuses on characterizing and modeling cave/paleokarst networks in reservoir formations. By using MPS, spatial distribution of cave passages in modern and ancient cave systems are described. But before describing the details of the modeling approach, the geological background of paleokarst systems, mechanisms leading to the formation of caves, associated facies descriptions and an overview of paleokarst hydrocarbon resources are provided in this section.

2.1.1 Geological Overview of Karst Systems

Karst systems are formed as a result of subaerial exposure of carbonate rocks. They are generally formed by a combination of processes such as dissolution, precipitation, erosion, sedimentation and collapse of landforms. Karstification is generally initiated along flow paths such as joints, fractures or along bedding planes and the host rock is generally altered by flowing fluids. Meteoric waters, aquifers, sea water, hydrothermal and underground fluids may cause initiation and development of karst systems.

In the early stages of karst development, pre-existing rock should either be exposed to surface or have permeable pathways for fluid flow. Permeable pathways within the host rock mainly follow primary depositional pore space, syn-sedimentary pores and tectonic fractures. Primary depositional voids can be observed mainly in reefal environments and they can be easily identified with the irregular outlines of the deposit (Smart et al., 1988). The syn-sedimentary voids are generally fractures developed by sedimentation and erosion processes like compaction, unloading and mass movement (Smart et al., 1988). Tectonic fractures are formed by tectonic movements and they strongly follow certain orientations (Smart et al., 1988).

Stages of karst evolution are graphically illustrated by Esteban and Wilson (1993) in Figure 2.1. Esteban and Wilson (1993) stated that during the initial and youth stages of karst evolution, pore space within the pre-karst formation is enhanced by either dissolution or erosion or both. At the initial stage, void space is developed by dissolution generally caused by mixing of different waters. At this stage, fluid flow within the pore space is laminar and erosion due to flow is not significant. During the youth stage, turbulent flow becomes dominant and results in an increase in porosity enhancement. According to Esteban and Wilson (1993), karst system reaches its mature stage when hydrological zones and lithological profiles are well established. At the late stage of karst evolution, porosity destruction due to cementation, sedimentation and collapse, is observed rather than porosity development.

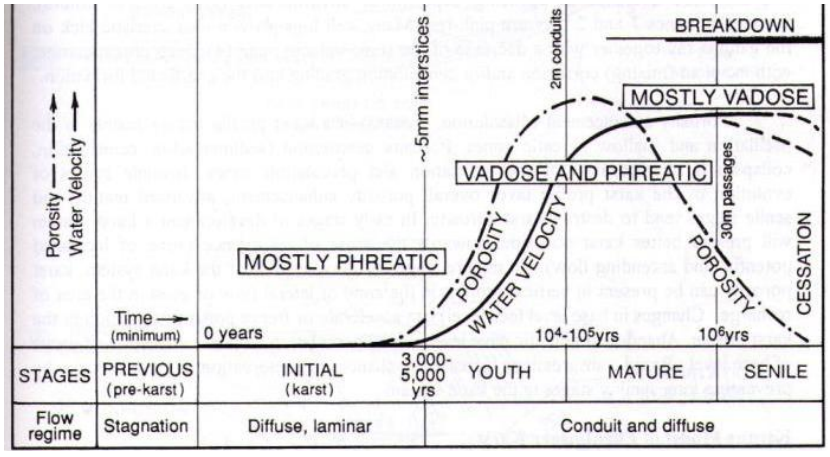


Figure 2.1- Stages of Karst Evolution (Esteban and Wilson, 1993)

Dissolution and erosion are the main mechanisms of pore evolution in karst systems. These processes are mostly effective in oscillation and shallow phreatic zones (Esteban and Wilson, 1993). On the other hand, extensive porosity destruction is observed in infiltration and percolation zones (Esteban and Wilson, 1993). Preservation

and destruction of porosity also depend on water table position. Thus, porosity of the karst system can be either enhanced or destroyed by the fluctuations in water-table position.

2.1.1.1 Karst Formation Mechanisms

Karsts are broadly divided into two main groups as epigenic and hypogenic karsts. In the formation of epigenic karsts, meteoric waters are the major cause of dissolution, whereas hypogenic karsts are generally formed by groundwater or hydrothermal fluids.

Epigenic karsts are formed by fluid flow in overlying or adjacent formations in recharge zones and continuous dissolution may yield collapse and formation of features like sinkholes (Palmer, 1991). As illustrated in Figure 2.2, epigenic karsts are formed by meteoric waters and continuous dissolution may result in collapse.

On the other hand, hypogenic karsts are formed by dissolution of host rock by deep acidic waters or fluid movements caused by deep subsurface activities such as H_2S flow, H_2S mixing with O_2-rich groundwater and migration of thermal-baric waters (Palmer, 1991). During the formation of hypogenic karsts, there is no connection to the overlying surface.

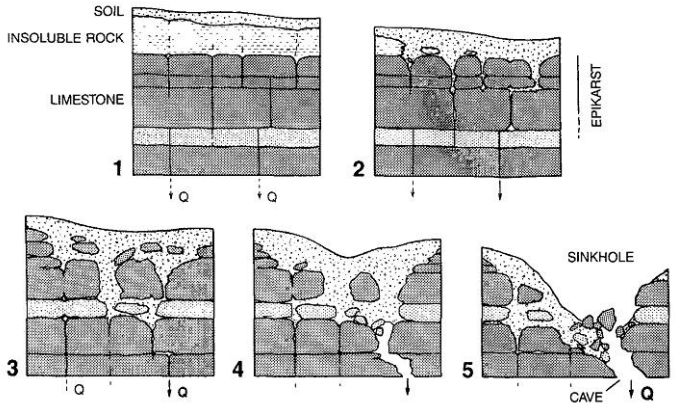


Figure 2.2- Evolution of epigenic karst and typical sinkhole. Epigenic karsts are formed by fluid flow in overlying or adjacent formations (Palmer, 1991)

Karst formations produced by the mixing of sea water and meteoric waters are called flank-margin caves or Island Karst Model. Mixing can be observed both at the bottom of fresh water lens and at the vadose-phreatic interface (Labourdette et al., 2007). Flank-margin caves can be either epigenic or hypogenic in origin. If mixing of meteoric and sea water is effective in dissolution, flank-margin cave is considered as epigenic. A cave formed by mixing of fresh-water aquifer and sea water is termed hypogenic because they are not connected to the surface hydrology (Labourdette et al., 2007). Although flank-margin caves are formed in an active depositional environment, simple subsidence or small changes in water level may yield cave preservation whereas surface erosion can be observed only after the cave formation (Ford, 1988). As reported by Ford, they can also be altered by collapse due to removal of debris by continuous and extensive dissolution (Labourdette et al., 2007). Flank-margin caves are also named as Carbonate Island Karst and corresponding profile as illustrated by Mylroie and Carew (Figure 2.3). Development of flank-margin caves is significantly affected by the location of fresh water-lens, sea level and the extent of the mixing zone. After the formation of cave margin, erosion due to the meteoric waters can be observed.

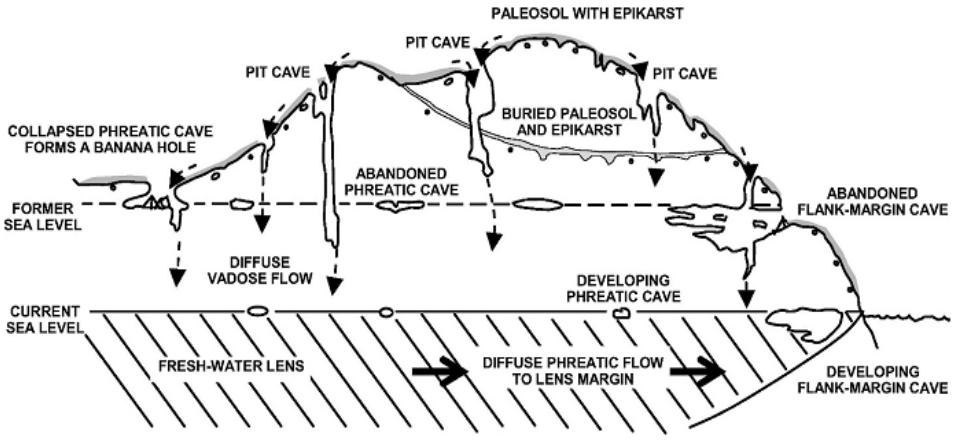


Figure 2.3- Carbonate Island Karst profile (Labourdette et al., 2007)

2.1.1.2 Factors Affecting Karst Formation

Karst formation is the culmination of many processes and thus depends on several factors. Choquette and James (1988) divided the factors affecting karst formation into two classes: intrinsic and extrinsic factors.

According to Choquette and James (1988), intrinsic factors affecting karst formation are mainly general lithology, matrix (stratal) permeability and availability of permeable pathways such as fractures or potential conduits. Maturity of the host rock is also an important factor in karst formation. Since immature rocks are altered easily compared to more mature formations, degree of karstification is enhanced in immature host rocks.

Stratal permeability and open fractures significantly affect fluid flow. In case of high stratal permeability with poorly cemented carbonate, groundwater flow will diffuse in the grain framework resulting in dissolution (Choquette and James, 1988). On the other hand, in case of low stratal permeability, fluid flow mostly occurs through the fractures and along bedding planes. Therefore, the extent of karstification is mostly affected by stratal permeability and permeable flow paths within the host rock.

In limestone formations, the degree of chemical stability and maturity, determines the solubility of the rock. Thus, porosity enhancement depends on the rock mineralogy and presence of soluble cement in the pore space (Choquette and James, 1988). Moreover, intrinsic factors such as fabric and texture of the rocks, bedding thickness and bulk purity mostly affect the dissolution and thus formation of karsts.

Extrinsic factors such as climate, base level elevation, vegetation, and time duration of exposure have significant effect on karst formation. The first important extrinsic factor is climate. According to Choquette and James (1988), in warm temperature areas with high rainfall, soil, sinkholes and subsurface dissolution collapse breccias are commonly observed. In temperate climates, karstification depends on seasonal or long term cycles. In arid environments, karstification is rather rare whereas in colder climates, karst formation is common despite the slow reaction rates within the formation.

The base level of the water table also significantly affects karst formation. Since karst landscapes can be eroded up to the water table level, base level controls the depth of surface erosion and maximum cave development level especially in epigenic karsts. Besides the aforementioned extrinsic factors, CO_2 or H_2S influxes out of water also control the dissolution and thus karst formation. Dissolution of $CaCO_3$ increases as the partial pressure of CO_2 dissolved in water increases and this process yields in porosity enhancement.

Furthermore, karst formation is also affected by H_2S influx in or out of ground water. According to Wright (2002), sulphuric acid produced by oxidation in deeper subsurface reacts with $CaCO_3$ to produce gypsum which is later dissolved. As it is stated by Hill, this process might yield sulphuric acid oil fields in karst formations (Wright, 2002).

Spatial distribution of karst structures are affected by the extrinsic and intrinsic factors. Therefore, understanding the properties of host rock such as permeability and degree of maturity, and effective extrinsic factors like paleo-climate conditions and duration of surface exposure might yield valuable information on spatial distribution of the karst network. Geologic evidences on changes in paleo-climate conditions, water level oscillations and presence of unconformity surfaces provide insights into the karst forming mechanisms and these observations might be used for identification of karst structures and prediction of inaccessible passages. Thus, recognition of a karst/cave system requires integration of various information on karstification mechanisms, properties of the host rock and conditions during karst formation.

2.1.2 Paleokarst Reservoirs

Modern day karst systems usually host aquifers with inlet and discharge locations. Thus, most of the studies on modern day karsts are in hydrology and they usually investigate evolution of karst systems, groundwater flow movements and distribution of karst conduits (Hill and Martin, 2008; Filipponi et al., 2009). On the other hand, petroleum reservoirs exist in paleokarst structures which are defined as ancient karst that are buried by the overlying formations (Choquette and James, 1988). Moreover, studies in paleokarst reservoirs mostly focus on recognition of paleokarst facies and generally do not examine the spatial distribution of these complex structures.

In modern systems, surface landform such exposed caves and discharge features like springs indicate the presence of karsts. In case of paleokarst reservoirs, it is challenging to recognize the paleokarst system during the early stages of field development. Therefore, several paleokarst features can be used for identification of paleokarst reservoirs.

According to Choquette and James (1988), paleokarst features are mainly divided into three groups; stratigraphic-geomorphic, macroscopic and microscopic. Stratigraphic-geomorphic features are mostly defined by karst landforms such as towers, dolines, closed depressions; unconformities and shallowing upward cycles which end abruptly at paleokarst surfaces. The second group is macroscopic structures which are classified as surface and subsurface karst features (Choquette and James, 1988). Some of the surface karst features are caliche, boxwork structures and non-sedimentary breccias. Caves, non-selective dissolutional voids, in-place brecciated and fractured strata, collapse structures, dissolution-enlarged fractures, rubble and fissure fabric, and breccias in irregular bodies are considered as subsurface macroscopic karst features. The third group of features are the microscopic structures which are defined by eluviated soil in small pores, etched, meniscus, pendant and needle-fiber vadose carbonate cements, and extensive, dissolution enlarged fabric selective pores (Choquette and James, 1988). By recognition of these features, paleokarst reservoirs can be identified in early stages of field development. For example, the Yates Field of West Texas is a paleokarst reservoir and its complex nature was recognized in early years of production by identification of several macroscopic and microscopic karst features (Levine et al., 2002).

Paleokarsts are developed at several locations depending on stratigraphical settings and are classified as depositional, local and interregional karsts in terms of stratigraphy (Choquette and James, 1988). In Figure 2.4, stratigraphy of paleokarsts is illustrated. Depositional karsts are formed within a depositional sequence and mostly are at the meter scale. They are formed by sediment accretion at or around sea level and are not significantly affected by surface exposure, near surface cementation and minor subsurface dissolution.

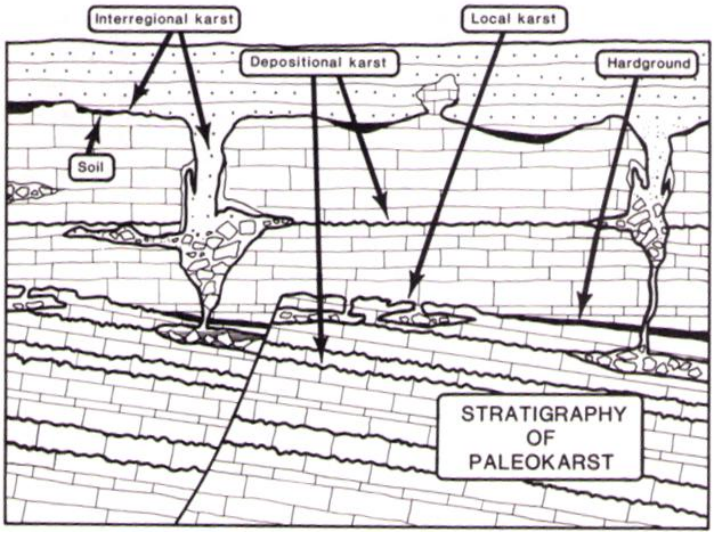


Figure 2.4- Stratigraphy of Paleokarst (Choquette and James, 1988)

Local paleokarsts are formed during the partial exposure of carbonate shelf or platform due to tectonism, small drops in sea level or syn-sedimentary block faulting. Effects of surface exposure on local paleokarst formation depend on the duration of exposure. Extent of local paleokarsts varies from cm scale to larger scales depending on the cycle duration. Interregional paleokarsts are formed by major eustatic-tectonic events and contrary to local paleokarsts, they are widely spread over a region. Since interregional paleokarsts are significantly affected by surface exposure, they undergo deep, pervasive dissolution and have distinctive karst features (Choquette and James, 1988).

2.1.2.1 Paleokarst Features

Burial and collapse are important steps of paleokarst and paleocave evolution. As dissolution of host rock progresses, stress on the cave ceilings and walls increases due to overlying formations. Fluids within the cave and transported sediments support the cave ceiling and walls against the stress. As burial continues and support of the ceiling span

decreases, collapse of the cave is inevitable. Thus, “stress is relieved by collapse of the rock mass within the stress zone” (Loucks, 1999) yielding a highly fractured and heterogeneous geologic feature with different facies. In Figure 2.5, evolution and burial of a near surface cave are illustrated (Loucks, 1999).

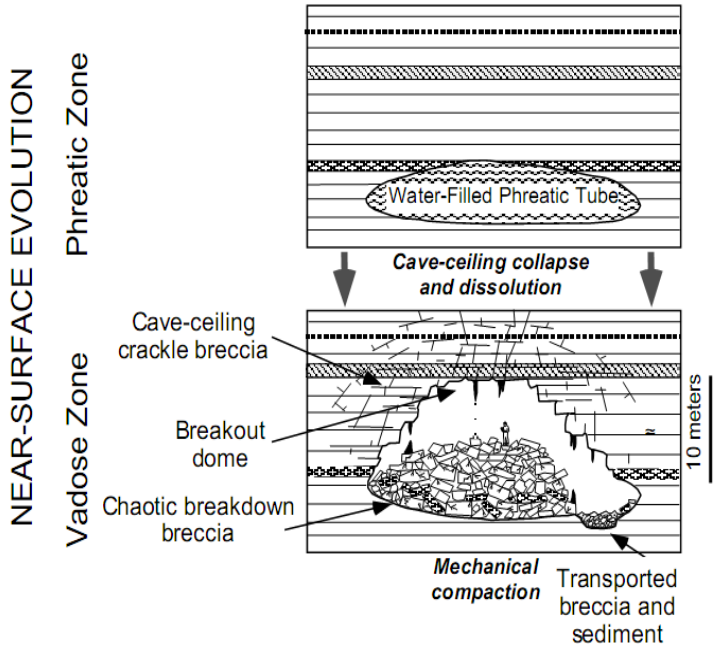


Figure 2.5- Evolution and Burial of Near-Surface Caves (Loucks, 1999)

Cave collapse results in paleocave facies consist of different sediments which are mostly fractured. Loucks (1999) proposed a ternary classification for paleocave sediments (Figure 2.6).

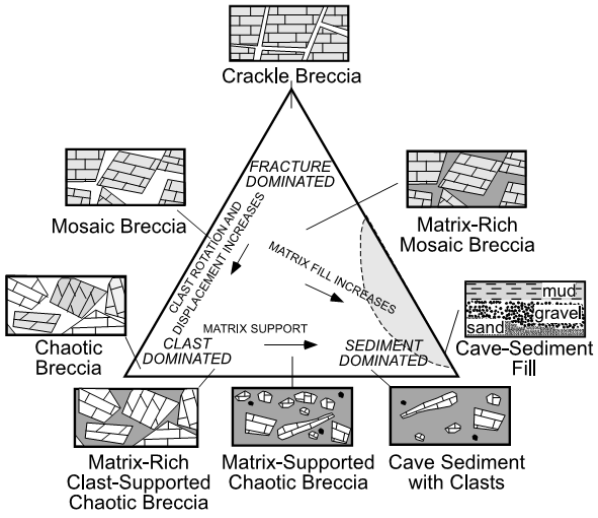


Figure 2.6- Classification of Cave Sediments and Breccias (Loucks, 1999)

According to Loucks (1999), cave sediments can be represented by three end members; crackle breccia, chaotic breccia and cave sediment fill. Crackle breccia is formed as a result of extensive fracturation and clast segments are separated by thin fractures (Loucks and Mescher, 2001). Crackle breccias are mostly found in cave ceilings and disturbed host rock where fracturing is mostly dominant. Mosaic breccias are similar to crackle breccias but they are close to the collapse zone of the cave ceiling. They are affected by displacement and rotation. Second end member is chaotic breccias which are product of cave collapse. They are derived from the host rock, mostly from the ceiling and cave walls during the collapse and then, transported to the cave base. Chaotic breccias are derived from different host rocks (Loucks and Mescher, 2001). Third end of the ternary classification is sediment dominated clasts. They are transported and deposited before the cave collapse, and are thus products of cave forming mechanisms. Sediments can be either from inside or outside of the cave and might be carbonate or siliciclastic with breakdown-material (Loucks, 1999). The size of sediments can range from clay to boulder, depends on the transporting mechanism. Sediment fill can be

observed at the cave base within a combination of mostly chaotic and crackle breccias. Porosity and permeability of cave sediments decrease with increasing cement and matrix content and fractures. Thus, crackle and chaotic breccia sediments have porosity in 2-20% range whereas matrix supported sediments has lower porosity varying from 1-5% (Kerans, 1988).

Loucks and Mescher (2001) divided paleocave facies into six basic classes and the classification is given in Figure 2.7. According to Loucks and Mescher (2001), each facies can be distinctly separated from the adjacent facies and in case of coalesced-collapsed systems, they may change gradually. The cave facies are classified as; undisturbed, disturbed, highly disturbed strata, coarse-clast chaotic breccias, fine clast chaotic breccias and sediment-fill.

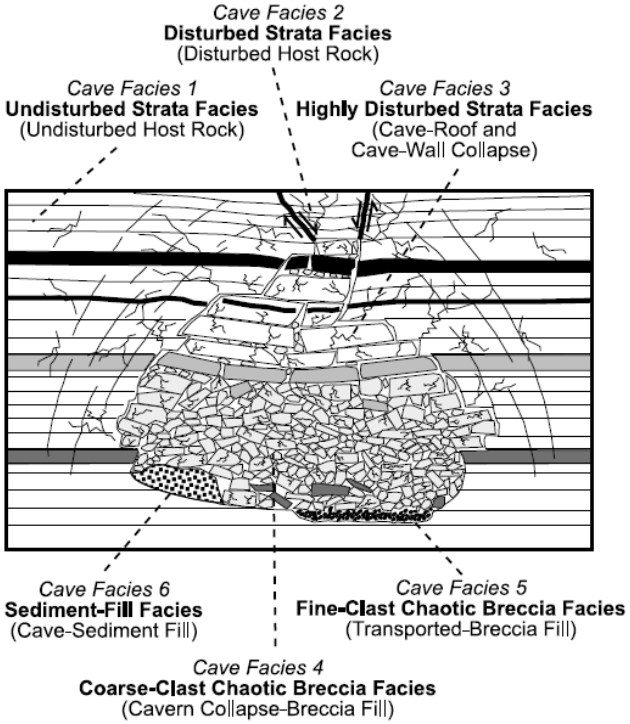


Figure 2.7- Classification of Paleocave Facies (Loucks and Mescher, 2001)

- Cave Facies 1: Undisturbed strata are composed of host rock and they show excellent bedding continuity with an aerial extent of tens to hundreds of meters depending on the depositional setting. The undisturbed strata generally have minor matrix and fracture porosity ranging from 3-5 % and permeability of the order of a few millidarcies.
- Cave Facies 2: Disturbed strata are mostly affected by collapse of cave structures and mostly contain crackle and mosaic breccias with significant amount of fractures. Due to fracturing, permeability is higher and usually in tens of millidarcies.
- Cave Facies 3: Highly disturbed strata are composed of collapsed rocks derived from host rock, cave roof and walls. This facies is discontinuous and consists of pockets and layers of chaotic breccias. Porosity ranges from 5 to 15 % and permeability varies from tens to hundreds of millidarcies with small scale folding and faulting.
- Cave Facies 4: Coarse chaotic breccia consists of collapsed clasts which are poorly sorted with varying sizes from granule to boulder clasts. The sediments are generally matrix supported, thus interclast pores mostly contribute to porosity. Porosity usually exceeds 20% and permeability is relatively high in the order of several darcies.
- Cave Facies 5: Fine chaotic breccia is formed by clasts of various sizes from granule to cobble that are transported from other regions in the karst system and moderately sorted. Similar to the coarse chaotic facies, fine chaotic breccias also have high porosity (greater than 20%) and high permeability.
- Cave Facies 6: Sediment fill facies is the last group of paleocave sediments identified by Loucks and Mescher (2001). It is composed of cave sediments that are commonly carbonate and/or siliciclastic debris transported from mostly outside the cave. The debris ranges from clay to cobble that may be poorly to well sorted. Sedimentary structures like cross bedding, fining or coarsening sequences, chaotic bedding or

lamination may be observed. Petrophysical properties of the sediment fill depend on the texture and the mineralogy, thus siliciclastic fills are typically tight whereas carbonate fill might be permeable (Loucks and Mescher, 2001).

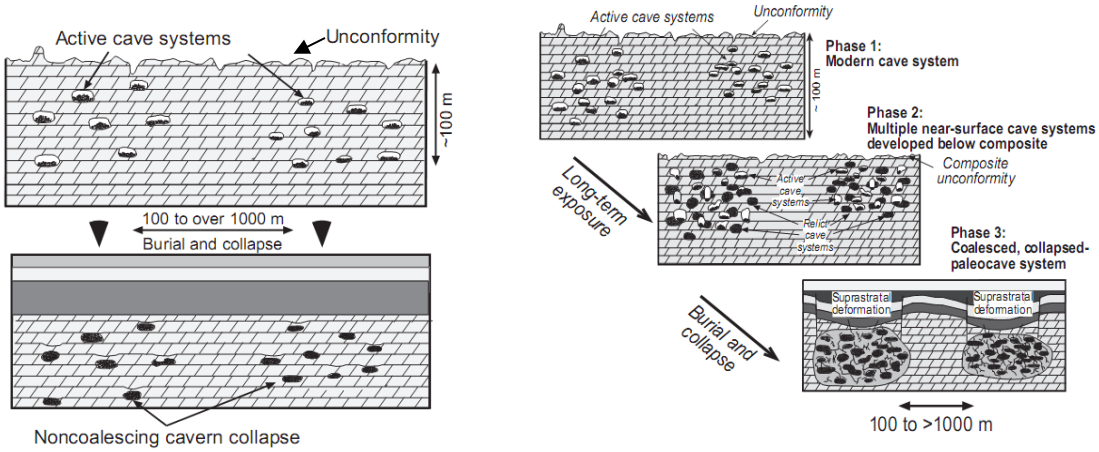
As illustrated in Figure 2.7, paleokarst reservoirs have a variety of facies with several heterogeneity features. These complex structures cause difficulties in identification of petrophysical properties as well as problems in drilling operations such as loss circulation and bit drops.

2.1.2.2 Paleokarst Systems

Several mechanisms and factors influence karst formation and evolution. Thus, identification and description of paleokarst systems require integration of those influencing factors as well as regional geology. Loucks (1999) divided paleokarst systems into two main groups as non-coalescing and coalescing based on the intrinsic and extrinsic factors affecting karst formation, petrophysical properties and geological history of host rock. The non-coalescing karst system consists of individual caves whereas the coalescing karsts are formed by coalesce and collapse of several caves.

Non-coalescing paleocave systems are composed of several individual collapsed caves and thus the structure is relatively simpler compared to coalescing systems. Individual collapsed caves usually have characteristic cave features and the collapsed paleocave structures are not deeply buried in comparison to coalescing systems (Figure 2.8a).

Loucks (2007) stated that karst networks with high areal density of passages with long surface exposure will coalesce and form coalescing systems (Figure 2.8b). Also, multiple surface exposures may yield coalesced-collapsed paleocave structures (Loucks, 1999). The coalescing systems usually have large areal extent of hundreds to several thousands of meters.



a- Non-Coalescing System b- Coalescing System

Figure 2.8- Paleocave Systems (Loucks, 2007)

Formation of coalescing paleocave systems is illustrated in Figure 2.8b. According to Loucks (2007), coalescing paleocave systems are composed of two main strata; a lower section composed of collapsed and coalesced caves and the upper section with the deformation features result of collapse and compaction. Fractures are observed around both the individual collapsed caves and the whole coalescence area. In order for coalescence to be initiated, a continuous path between the paleocave structures should exist (Loucks, 1999). The fractures and fractured breccias of the collapsed features are considered to be the pathways.

The upper suprastratal (sag) deformations are composed of subsidence area with large scale fault and non-tectonic fractures (Loucks, 2007). The subsidence area corresponds to collapsed zone of the coalesced cave system. Sag deformations can be bounded by circular faults that extend over large areas. Due to the extent of overburden required to collapse several caves in a region, the coalesced zone of paleokarst structures is generally deeply buried. For petroleum exploration purposes, recognition of coalescing

paleocave systems can result in large exploration target areas with significant pay zone thicknesses.

2.1.2.3 Recognition of Paleokarst Systems in Subsurface

The recognition and characterization of reservoir properties require additional effort because of the heterogeneities in carbonate rocks. Syn-deposition and post-deposition processes in paleokarst reservoirs produce complex facies and extremely heterogeneous structures that are described in the previous sections. Thus, identification of a paleocave system and predicting its areal extent are essential for accurate reservoir forecasting. Paleocave systems can be identified by integrating various features at field and well scales.

Karst features identified at field scale are important evidences for the presence of paleocave in subsurface. Sinkholes, sag deformations, outcrops, unconformity features and surface caves/karst topologies are field scale evidences of paleocave presence. Sinkholes are breccia pipes formed by dissolution and collapse of formations by meteoric waters. These surface landforms directly indicate presence of a subsurface paleokarst. Besides sinkholes, sag deformations are developed by collapse of subsurface caves. Since they are related to coalesced-collapsed paleocave systems, suprastratal deformations are considered as evidence of paleocave. Moreover, outcrops and unconformity features such as karst breccias, unconformity surfaces, caliche features are products of surface exposure. These features can be used for relating regional events to global events by interpreting unconformity surfaces that support presence of paleocave structures.

Several well scale data such as drilling records, well logs, core samples, well tests and production profiles, are integrated in order to identify paleocave intervals along the wellbore. Drilling records provide direct evidences for presence of paleocaves. Sudden bit drops and unexpected lost circulation zones are usually observed while drilling

through paleocave structures. Well logs are commonly used for identifying paleocave intervals in hydrocarbon reservoirs. According to Tinker et al. (1995), paleocave zones are identified by examining caliper, GR and bulk density logs. Caliper logs show excursion from baseline in cave intervals whereas, GR values less than 40 API and bulk densities less than 2 g/cc indicate cave zone (Tinker et al., 1995). Moreover, neutron porosity logs are also used for identification of these complex structures (Dembicki and Machel, 1996). Paleocave intervals are described by high neutron porosity values with low GR and bulk densities.

In addition to well logs, paleocave features are also identified using core samples. Core intervals with crackled and chaotic breccias, solution enlarged vugs and fractures indicate presence of paleocave structures. Well test data and production profiles are also used for recognition of paleocave reservoirs. For example, confined karst systems may behave like dual porosity systems with a high initial oil production from high permeability-high porosity cave interval followed by lower production rates from low permeability carbonate matrix. In this type of systems, the host rock has low permeability with little or negligible interparticle porosity and dissolution mostly occurs in solution-enlarged bedding planes, fractures and joints Meyers (1988). On the other hand, diffuse karsts with high permeable host rocks, are relatively homogeneous due to the lack of collapse features. Since dissolution processes are not restricted to the bedding planes, fractures and joints in diffuse systems, solution enlarged fractures and vugs are commonly observed and the formation act as a homogenous media with high porosity and high permeability Meyers (1988).

In conclusion, paleokarst systems are recognized by careful integration of geologic studies, detailed facies description, well logs, drilling, production and well test

data. It is essential to recognize paleocave features and implement them in reservoir modeling studies for accurate reservoir description and accurate future predictions.

2.1.2.4 Important Paleokarst Hydrocarbon Reservoirs

Some important hydrocarbon resources are present in paleokarst reservoirs such as the Ghawar Field in Saudi Arabia, the Ordos Basin of China, the Yates Field in West Texas, the Grosmont Formation in Canada and the Kirkuk Field in Iraq.

The Ghawar Field in Saudi Arabia is the world's largest oil reservoir with an oil production rate of 5 MMB/D from Jurassic Arab-D reservoirs and a gas production rate of 8 billion B/D from Paleozoic reservoirs (Afifi, 2005). The Arab-D formation has three different dolomite types; fabric preserving (FP), baroque and non-fabric preserving (NFP) (Cantrell et al., 2004). Fabric preserving dolomite associated with the Arab-D anhydrite occurs as laterally discontinuous thin layers (Cantrell et al., 2004). Baroque dolomite which is non-fabric preserving, is found as vertically pervasive, thick layers and aerially restricted. This type of dolomite mostly consists of coarse and saddle-shape crystals (Cantrell et al., 2004).

Non-fabric preserving (NFP) dolomite is most dominantly present in the reservoir. It is observed throughout the Arab-D reservoir spread over tens of square kilometers (Cantrell et al., 2004). Since NFP dolomite is extensively altered, intercrystalline porosity is abundant and is locally responsible for super-k zones in the reservoir (Cantrell et al., 2004). Super-k zones yield high productivity from confined layers of 10-20 feet (Al Shahri and Al Muraikhi, 1998, Dogru et al., 2001). According to Dogru et al. (2001), the super-k zones consist of vugs, faults, fractures and stratiform dolomites and these high permeability zones behave like high velocity flow conduits.

The model of dolomitization for the Arab-D reservoir in the Ghawar Field was presented by Cantrell et al. (2004) who conclude that Super-k zones are the product of a

hypogenic karst system extending throughout the entire field with a major trend following the fracture/fault network system (Figure 2.9).

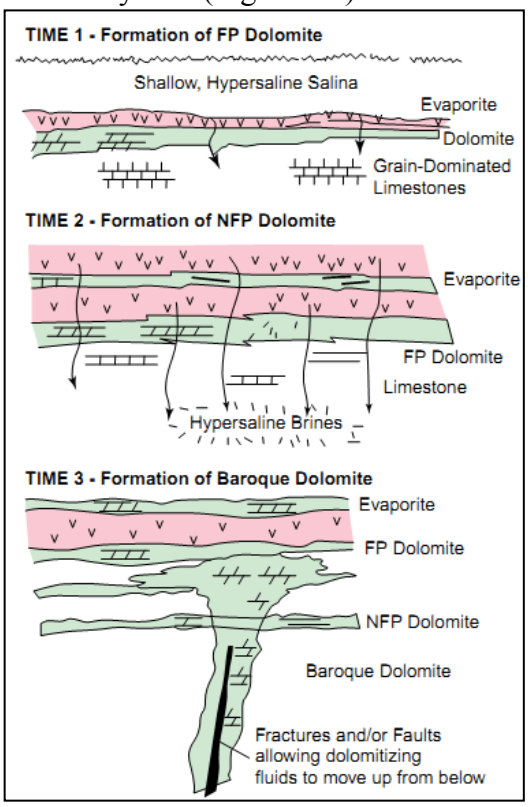


Figure 2.9- Model of Dolomitization for Arab-D Reservoir in the Ghawar Field (Cantrell et al., 2004)

The Ordos Basin is the second largest sedimentary basin in China with several giant gas fields. The Jiangbian gas field is the largest of them with a proven initial gas in place of 11 trillion cubic meters and it is a Lower Ordovician paleokarst reservoir (Li et al., 2008). The Jiangbian Field was discovered in 1987 and field appraisal continued until 1995 and Li et al. (2008) reported that the main reservoir member of the Jiangbian field consists of finely crystalline limestone, dolostone and karst breccia. The reservoir dolomite accounts for 85% of the total pay zone of 30-90 m with porosity ranging in 2.53-15.2% (average porosity is 6.2%) and permeability varies from 0.0128 mD to 1050

mD (average permeability is 2.66 mD). Pore space is mainly composed of dissolution pores, cavities, intercrystalline and intergranular dissolution pores, and moldic porosity. Moreover, fractures are also effective for fluid flow and most effective fractures are related to the karst structures (Li et al., 2008).

In Ordos Basin, the paleokarst system was formed at the end of Ordovician due to a regional uplift and long term surface exposure. The reservoir is mainly controlled by karst paleo-geomorphology and diagenesis. Strong karstification is observed in eastern side of the field and the dissolution is mostly caused by horizontal movement of ground water.

Another important oil reservoir in a paleokarst setting is Yates Field in West Texas which was discovered in 1926. The original oil place is 5 billion bbl and the recovery factor is reported as 29%. The main reservoir is Permian age San Andres formation carbonates which is 96% dolomite in the eastern portions of the reservoir (Tinker et al., 1995). Reservoir rock quality degrades with increasing shale content towards the western regions of the Yates Field. Several studies were conducted for detailed description of the formation and identification of diagenetic events in the Yates Field (Spencer and Warren, 1986; Craig et al., 1986; Tinker et al., 1995). Multiple unconformity surfaces were observed and extensive karstification was identified. It was determined that three cave forming events occurred in the Permian San Andres Formation due to the mixing zone dissolution. Subaerial exposure and karstification resulted in a network-maze cave pattern that is controlled by pre-existing joints in the field. The paleokarst system is dominant in the East side of the reservoir. Further details of the Yates Field and the distribution of paleokarst facies will be given in subsequent chapters of the dissertation.

Paleokarst reservoirs are also major heavy oil hydrocarbon resources. The Upper Devonian Grosmont Formation in northeastern Alberta, Canada is a paleokarst reservoir with immense reserves of 7° API bitumen (approximately 315 billion bbl) (Dembicki and Machel, 1996). The reservoir was formed in a shallow marine carbonate platform which was subaerially exposed between the Mississippian and Cretaceous age. Extensive karstification resulted in irregular erosion surface, meter-size dissolution cavities, collapse breccias, sinkholes, paleosols and fractures in the Grosmont formation (Dembicki and Machel, 1996).

Petrophysical properties, reservoir thickness and seal effectiveness are significantly affected by karstification in the Grosmont formation. Porosity and permeability are either enhanced or degraded; enhanced porosity can be more than 40% and permeability up to 30 D. On the other hand, reservoir properties can be degraded due to calcite cementation or infiltration of cretaceous sands and clays. According to Dembicki and Machel (1996), porosity is highly heterogeneous in the Grosmont formation. Early dolomitization can result in intercrystalline porosity of 20% and extensive karstification can increase it up to 40% and locally up to 100% (meter size caverns). Permeability of the dolostone zones ranges from 10 to 100 mD in non-karsted intervals whereas in regions with extensive karstification permeability up to 30 D can be observed. Dembicki and Machel (1996) stated that dissolution of the Devonian strata formed epigenic cave network in the Grosmont formation, and consequent cave collapse resulted in vertical to subvertical fractures and crackle breccias. They concluded that the Grosmont formation is comparable to the San Andres formation of Yates Field.

Trice (2005) presented a review of 44 reservoirs in Middle East and performed comparative analyses based on several reservoir properties such as reservoir geology and production profile. According to Trice (2005), Kirkuk Field which is located in northern

Iraq and was discovered in 1927, is classified as a paleokarst reservoir. Main producing zone is the main limestone of the Eocene-Oligocene fractured carbonate formation and the total production has been 9.1 BBO till 1980 (Trice, 2005). The main limestone zone mostly consists of dolomitized skeletal packstones, grainstones and rudstones (Trice, 2005). Porosity and permeability vary throughout the field; backreef and reef facies have porosity in the 0-10 % range and permeability of 0-5 mD (Daniel, 1954). On the other hand, fore-reef facies which are altered by dissolution have porosity values ranging from 18% to 36% and permeability of 50-1,000 mD (Daniel, 1954).

The main limestone zone has three carbonate deposition cycles and subaerial exposure at the end of each cycle resulted in karstification and dissolution caverns. Daniel (1954) stated that almost vertical caverns were formed by dissolution along pre-existing joints whereas caves of mainly lateral extent were developed due to movement of meteoric waters along the pathways of the strata. According to Trice (2005), dissolution was dominant in the foreslope and platform margin buildup facies, and it was significant in the fore-reef and bank-margin. Both studies on the Kirkuk field suggested that the dissolution features and the caverns were developed by meteoric waters flowing along preferential pathways of existing joints and permeable facies.

2.1.2.5 Current Approaches in Paleokarst Reservoir Modeling

Paleokarst reservoirs have been modeled by using high permeability multipliers in flow simulation models (Dogru et al., 2001), dual-porosity/dual-permeability formulations (Uba et al., 2007) and stochastic techniques considering stratigraphic layering and paleobathymetry curves (Massonat and Pernarcic, 2002).

Most of the reservoir simulation studies on paleokarst reservoirs have been conducted and documented by Saudi Aramco. These papers generally focus on understanding and integrating super-k structures of paleokarst facies in the Ghawar Field.

Dogru et al. (2001) simulated a portion of the reservoir by using a 2.5 million-cell reservoir model. They investigated those parts of the Ghawar Field with extremely conductive channels that consist of vugs, faults, fractures and stratiform dolomite rocks. A dual porosity grid was imposed to handle high permeability layers of 70 Darcies. The geologic model had 134 layers with super-k zones with a maximum permeability of 8 Darcies. The model had a uniform areal grid of 250*250 meters with vertical layer thickness varying from a fraction of a foot to 2.5-3 ft. The fracture grid was superimposed on a fine grid area with a 1 km spacing and width of 20 meters. Reservoir simulation and history matching studies were performed using a model with multi-million cells. Super-k formations were described by 67 vertical layers with thickness varying from 1-6ft. History match was successfully obtained by changing permeability from 40 to 70 Darcies in super-k regions.

Uba et al. (2007) applied Warren-Root formulation for modeling flow in fractures and the Super-k layer of the Arab-D reservoir. They reported that Super-k layers were interconnected with the fracture corridors thus forming a combined highly conductive porous medium. A dual porosity/dual permeability simulation model was constructed with a total of 9 million cells. Super-k layers were represented as horizontal fractures of limited extent (Figure 2.10). History match was successfully obtained by adjusting effective Warren-Root and fracture parameters.

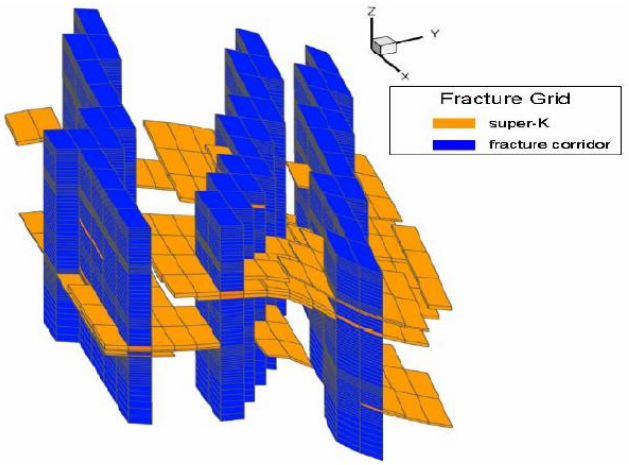


Figure 2.10- Fracture Grid for Simulation- Super-K and fracture corridors combination (Uba et al., 2007)

Besides describing paleokarst reservoirs using multi-million cells, Massonat and Pernarcic (2002) proposed a stochastic modeling approach. **Neptune** is a tool developed as a GOCAD plug-in by TotalFinaElf. The algorithm is based on the relationship between depositional facies at the shelf marine environments and water depth, energy of deposit and the stratigraphic context. **Neptune** yields models with realistic and consistent stratigraphic layering that are conditioned to field specific information in the form of facies proportions computed from hard data. Once, depositional facies models are built using **Neptune**, probabilities of facies to be affected by early karstification is evaluated.

The **Neptune** algorithm follows a workflow that integrates water-depth logs, seismic data and paleo water-depth reference maps. At the initial step, paleobathymetry curves are obtained at the wells and correlations using accommodation potential curves are developed. The next step is construction of a 3D model by extrapolating the accommodation potential curves. Local subsidence portions of the accommodation potential curves are obtained by kriging, Gaussian simulation or a combination of the two methods conditioned to the well data. Stratigraphic and 3D paleobathymetry grids are also constructed by stochastic techniques. Then, facies proportion cube is obtained by

combining depositional facies and paleobathymetry. Facies simulation in **Neptune** is performed by using the facies proportion cube in a truncated Gaussian simulation (Xu and Journel, 1993; Le Ravalec-Dupin et al., 2004; Da Veiga and Le Ravalec, 2010). Sequential indicator simulation (Alabert et al., 1992) or Boolean techniques (Lanzarini et al., 1997) can also be used at this step. Simulation of petrophysical properties is conducted by using the simulated facies model and incorporation of diagenetic features. Finally, **Neptune** provides a 3D reservoir facies model which includes the regions affected by karstification and associated petrophysical properties. According to Massonat and Pernarcic (2002), the output model from **Neptune** yields accurate predictions of paleokarst facies distribution. The model can be implemented into reservoir simulation studies. However, the number and complexity of the data pre-processing steps are daunting and the procedure requires information about paleobathymetry, paleo-water table etc. that might not be readily available.

2.2 STOCHASTIC MODELING TECHNIQUES IN RESERVOIR CHARACTERIZATION

Stochastic modeling techniques have been widely used in reservoir characterization studies. Geostatistical techniques involving two-point and multiple-point statistics are practical tools to model hydrocarbon reservoirs and complex geological structures.

2.2.1 Variogram Based Geostatistical Modeling Techniques

Most common geostatistical techniques such as kriging and Sequential Gaussian Simulation (SGSIM) are constrained to two-point statistics such as the semi-variogram or covariance and require discretization of the reservoir using grids. Although algorithms like co-kriging (Goovaerts, 1998) enables the integration of different data, the major assumption of multivariate Gaussianity and the ability to constrain only to two-point

statistics prevent accurate reproduction of complex reservoir structures (Strebelle, 2002). Methods like SGSIM, indicator-based techniques and truncated Gaussian simulation algorithms are commonly used in reservoir modeling.

According to Alabert et al. (1992), SGSIM is generally implemented for modeling reservoirs that do not exhibit any continuity of extreme features such as fractures or baffles or in formations with limited geological/lithological information. This method is variogram-based and requires Gaussian data sets. Since variograms are not adequate for representing complex features, it is difficult to reproduce curvilinear geological features such as faults, fractures and fluvial channels in kriging or SGSIM (Strebelle, 2002). Indicator simulation algorithms are used for modeling both continuous and discrete properties and they are mostly implemented in facies description studies. At a particular location, the facies is represented by an indicator variable which is either 1 (if the facies is present) or 0 (if it is absent). Indicator variable information and spatial description are input to the model in the form of hard conditioning data, facies proportion and an indicator variogram. According to Alabert et al. (1992), indicator simulations can model complex heterogeneity patterns and different spatial correlation structures for each facies. They also point out that representing each indicator variable using individual variogram provides flexibility to handle complex structures. Despite the advantages of indicator simulation, it has major drawbacks such as the possibility of inconsistent indicator variograms that can cause probability constraints to be violated, low computation efficiency in case of multiple indicators and incapability of reproducing well-defined geometries (Alabert et al., 1992).

Truncated Gaussian Simulation (TGS) is a practical tool for facies modeling and it has been widely used in reservoir characterization studies (Xu and Journel, 1993; Le Ravalec-Dupin et al., 2004; Da Veiga and Le Ravalec, 2010; Biver et al., 2012).

Realizations of lithofacies are obtained by truncating a stationary Gaussian field using a correlated random field. The truncations yield discrete facies. According to Caceres et al. (2010), TGS is successful in modeling of formations with defined stratigraphical successions. This method requires a truncation rule that incorporates lithofacies proportions and original facies indicator data. Conditioning to static and dynamic data is performed using kriging and gradual deformation methods (Le Ravalec-Dupin et al., 2004). Although TGS yields fast and accurate results while reproducing the original indicator variograms and lithological contacts, it has some limitations. TGS requires a truncation rule based on facies distribution and facies contact relations. It is challenging to obtain this kind of information for complex reservoirs with limited data. Moreover, the simulation utilizes a unique covariance for the Gaussian field and this yields all simulated facies to share similar spatial characteristics (Xu and Journel, 1993).

The aforementioned geostatistical techniques are successful for modeling homogeneous reservoirs whose properties are accurately described by variograms. On the other hand, most carbonate reservoirs have complex geological structures such as fracture networks, facies distributions etc. As a result, geostatistical methods using higher order statistics have to be developed to characterize and model heterogeneous reservoirs.

2.2.2 Multiple-Point Statistics (MPS) Algorithms

Variogram-based geostatistical methods are incapable of successfully modeling complex geologic features and curvilinear geometries such as fluvial channels or natural fractures (Strebelle, 2002). For better representation of such features, multiple point statistics-based approaches that capture the joint spatial variability at 3 or more locations are required.

MPS basically depends on statistical relationships exhibited by groups of three or more data. Higher order statistics summarizes the spatial patterns exhibited by data

points. Unlike two-point statistics, a training image and a template are required in MPS and data variability is quantified in the form of a pattern histogram. Pattern histogram is constructed by scanning a training image using a template. The training image is obtained from outcrop analysis, geological maps or near well bore images and it transfers geologic heterogeneity by replacing the variogram (Caers and Zhang, 2004). According to Caers and Zhang (2004), training images are an explicit quantification of prior geologic heterogeneity. Once the training image is selected, templates are used for scanning and then constructing the pattern histogram. Template selection is critical; template size should be large enough to capture the patterns in the training image while yielding a computationally efficient scanning process.

In conventional MPS algorithms, training images, templates and simulation results are in form of grids (Strebelle 2002; Arpat and Caers, 2007) (Figure 2.11) and the training image, i.e. facies distributions, are usually in binary format. Training image should be selected or constructed carefully since it carries the overall geological structure information. Moreover, it should obey the rules of stationarity and ergodicity. Caers and Zhang (2004) stated that stationarity implies similarity and repetition of structures for calculating reliable statistics. Also, the training image should be large enough to avoid significant fluctuations in the statistics.

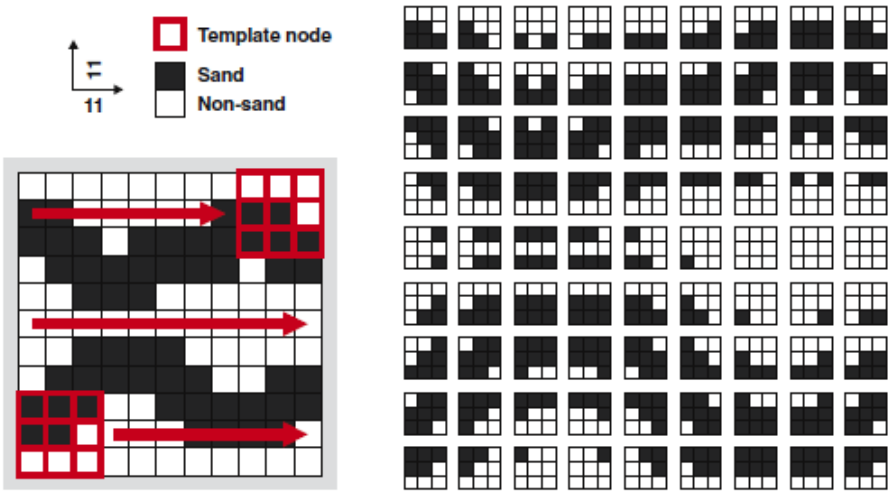


Figure 2.11- Example of Training Image and Its Patterns (Arpat and Caers, 2007)

Stochastic simulation studies constrained to realistic geology and involving MPS algorithms are relatively recent and mostly focused on description of geological structures and subsequent history matching (Caers et al., 2000; Strebelle, 2002; Caers, 2002; Caers and Zhang, 2004). An MPS simulation algorithm, *snesim (single normal equation simulation)*, which is based on the sequential simulation paradigm, was developed for conditional simulation of complex geological structures by Strebelle (2002). Recently, Arpat and Caers (2007) proposed *simpat, pattern-based simulation*, which aims at pattern reproduction instead of statistics reproduction while honoring conditioning data. The training image is segmented into the salient spatial patterns characterizing it and subsequently during simulation, an entire pattern is applied at a location depending on the local conditioning information at that location. This algorithm enables the use of multi-grids and can handle complex, discrete and continuous patterns at different scales.

MPS has been implemented for modeling geologic features in history matching studies (Caers et al., 2000; Caers, 2002; Eskandaridalvand, 2008). It is reported that MPS

is fast and robust, and superior to the traditional two point statistics while realistically reproducing the geology. Recently developed MPS algorithms are capable of representing complex geologic structures as well as integrating different data sets (Caers et al., 2000; Strebelle, 2002; Caers, 2002; Caers and Zhang, 2004). Moreover, curvilinear geological features of paleokarst reservoirs such as fractures can be represented accurately by using MPS (Liu and Srinivasan 2005; Liu et al., 2009).

The initial study involving a growth-based MPS simulation algorithm honoring conditioning data (Liu and Srinivasan 2004), was successful in reproduction of realistic fracture distributions observed in the training image of available field data. It was observed that accuracy and robustness of the simulation results mostly depend on fracture characteristics obtained from image logs and it was concluded that in case of having a representative training models of the fracture systems, MPS yields successful results while integrating geomechanical information (Liu and Srinivasan 2004).

Liu et al. (2009) presented an application of MPS for fracture network modeling that was then input to a thermal-hydrological-mechanical (THM) simulation of Yucca Mountain. Initially, they simulated fracture network of Yucca Mountain waste repository system by implementing a grid-based fracture network as a training image. The training image was constructed based on statistical properties of fracture spacing and orientation in Yucca Mountain. (Liu et al., 2009). They followed the pattern simulation algorithm proposed by Arpat and Caers (2007). The authors showed that MPS algorithms are successful for modeling natural fracture networks while reproducing connectivity of the system.

While MPS algorithms successfully represent complex geologic structures, in traditional implementations, a gridded training image is required for inferring the statistics. However, modern caves that are analogous to paleokarst formations are

represented by surveys reporting XYZ coordinates of cave central line. It is not practical to perform gridding and apply a rigid gridded template to calibrate statistics using such "point" data surveys. There are some recent studies focused on the determination of spatial distribution of cave size and stochastic modeling of cave formation (Henrion et al., 2008; Madriz, 2009), however, these studies do not address the problem of inference of pattern statistics from sparse data. Thus, a methodology for characterizing and modeling paleokarst reservoirs is required to provide better description of such complex facies in reservoir simulation studies for understanding flow mechanisms. In order to address these issues, we have developed an MPS algorithm that works on a non-gridded basis to characterize cave and paleokarst networks using "point-set" information. The algorithm basically consists of MPS analysis using flexible spatial templates and subsequent pattern simulation using the calibrated statistics. In Chapter 3, the proposed MPS algorithm and its applications on 2D and 3D data sets are presented.

Chapter 3: Non-Gridded MPS Analysis

In this chapter, a new multiple point statistical (MPS) analysis technique is presented to characterize and simulate connected geologic features such as cave networks and paleokarst reservoirs. The MPS technique uses non-gridded spatially flexible templates with various distance and angle tolerances instead of rigid gridded spatial templates that are used in conventional algorithms. The training image is also non-gridded; it consists of points and corresponding connectivity information. In this chapter, the method for inferring multiple point statistics using a non-gridded training image will be presented and its application on various synthetic and real field cases will be explained in detail.

3.1 NON-GRIDDED TEMPLATE

The spatial template has a crucial impact on calibrated statistics and pattern simulation in MPS. In this non-gridded approach, the spatial template is described by a group of points whose distance and angle configurations are defined with respect to a center node (Figure 3.1).

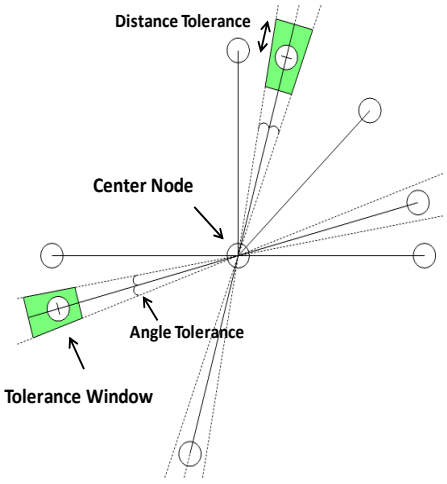


Figure 3.1- Spatially Flexible Non-Gridded Template with Tolerance Window

In order to compute the required multiple point statistics, distance and angle tolerances are specified while scanning the training image. Around each template node, a tolerance window is constructed using user-specified distance and angle tolerances. Distance tolerance values are specified as a fraction of the lag spacing between nodes of the template while angle tolerances are defined in a manner similar to variogram computation using irregularly spaced data. Specification of these tolerances results in a small tolerance window around the corresponding template node as shown in Figure 3.2. Points in the training image might not be aligned along the same exact line but with a slight deviation. Thus, tolerance window enables the overall pattern to be delineated despite small deviations.

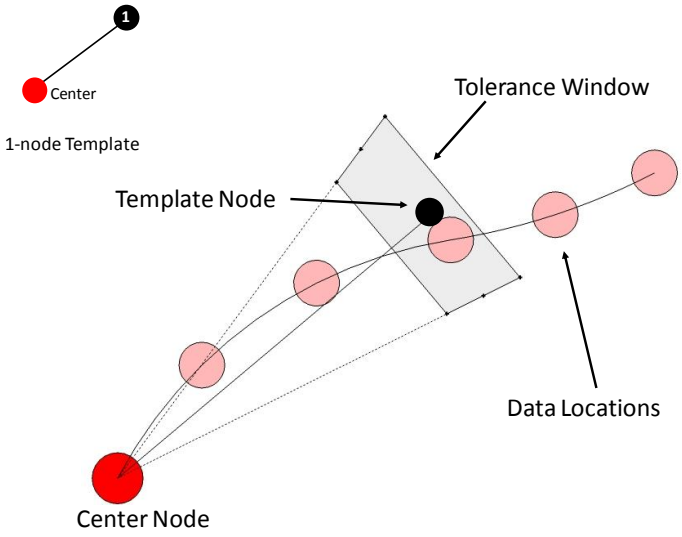
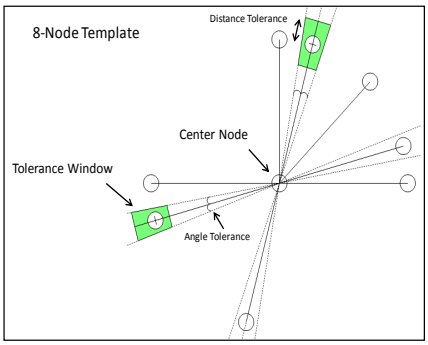


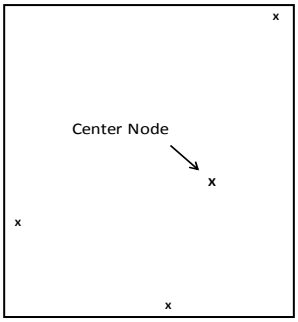
Figure 3.2 - Tolerance window demonstration using 1-node template.

3.2 MPS SCANNING ALGORITHM

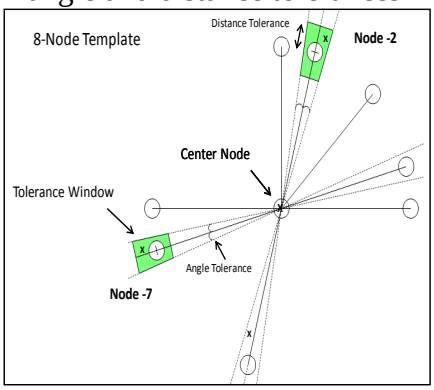
The non-gridded MPS analysis algorithm is illustrated in Figure 3.3. First, a spatial template is constructed using user specified lags, orientations and tolerances. Then, a data point from the point set training image is selected (Figure 3.3a-b).



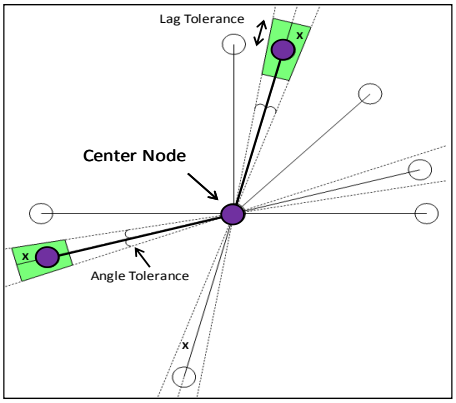
a. Define a template with angle and distance tolerances



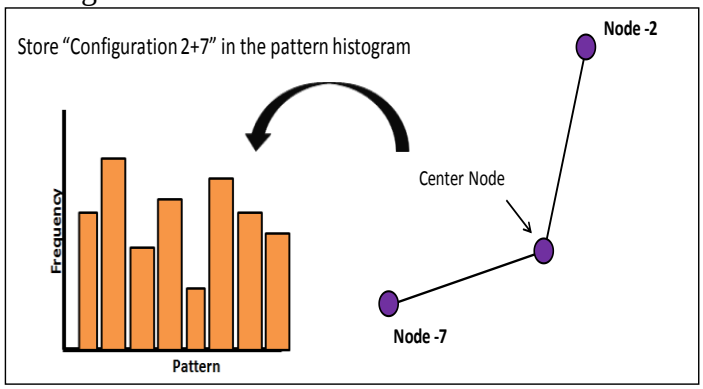
b. Select a node location from the data set.



c. Center the template around the selected node and scan current pattern of surrounding data.



d. Pattern of data observed at the current location of the template.



e. Update the pattern frequency by translating the template over the entire data set and compute the resultant pattern histogram.

Figure 3.3- Non-Gridded MPS Analysis Algorithm

The template is centered on the selected node and all surrounding data that fall within the tolerance window of any template nodes are determined (Figure 3.3c). If a data point falls into the tolerance window, the corresponding template node is activated and the resultant pattern is stored (Figure 3.3d). By translating the template over the entire training data set, frequencies of the observed patterns are continuously updated and ultimately the pattern histogram is constructed (Figure 3.3e). In some cases, the pattern statistics may be inferred using multiple templates (in lieu of one large, complex spatial template).

In the case of large point set training images (e.g. data sets including more than 10,000 points) and computation limitations, statistics are calculated by using a refined portion of the entire image. The size of the refined set is user defined and the original point set is refined by randomly selecting data locations. Once the pattern histogram is constructed, calibrated statistics are used in the pattern simulation of cave/fracture network.

3.3 MPS Analysis Using Generic Templates

Non-gridded MPS analysis is performed using spatially flexible templates. Template size is based on the length of connected passages in the network and overall network extent. Template configuration depends on the major cave network orientation. Thus, an initial assessment of the training image should be performed to identify network orientation and network extent prior to calculating MP-statistics. This can be accomplished by trying out some initial generic templates.

Generic templates are used for preliminary evaluation of major passage orientation and average passage length. Templates are refined and constructed by several steps. First, a set of generic templates is constructed mostly based on local geology and prior knowledge of the cave network. These templates are orientated along various dip

angles that are consistent with the cave orientation. The template nodes are placed at a succession of azimuth angles (Figure 3.4) spanning a 90^{o} range.

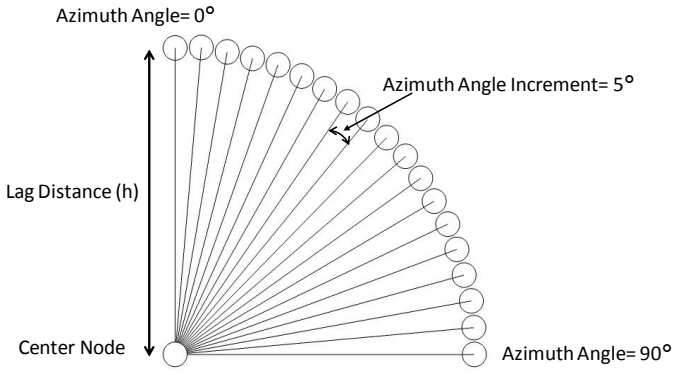


Figure 3.4- 2D Generic Template

After scanning the point-set training image, the number of connections captured by each generic template is compared and configurations that honor overall geology and have high number of connections are determined. Thus, major dip and azimuth directions of the network are identified. A second set of templates is constructed by fixing the main orientation and varying the template size. Template nodes are placed at various lag distances along the major orientation. Lag distances, i.e. template size, is increased sequentially in order to avoid capturing poor connections which are not physically meaningful. Then, lag distances at which a high number of connections are captured are identified. This procedure yields an optimum template size and identification of major cave orientation.

The use of generic templates provides a quick assessment of the network and the gathered statistics are used for constructing more complex templates for computing MP-statistics and pattern histogram. Details of MPS analysis using these complex templates are given in the next section.

3.4 MPS Analysis Using Complete Spatial Templates

Non-gridded MPS analysis of the point-set training image is performed by using a set of spatial templates that honor the network orientation and connectivity. Since generic templates provide an overview of the system in terms of major network orientation and length of connected features, expanded spatial templates are constructed by taking this information into account.

Spatial templates are defined such that they inherit the overall connectivity of the network. The templates usually consist of more than two nodes and lag distances are assigned to capture multi-scale features (Figure 3.5). Template nodes do not have to be symmetric around the center node; their configuration could be in various shapes like cross, star-shape, y-shape etc.

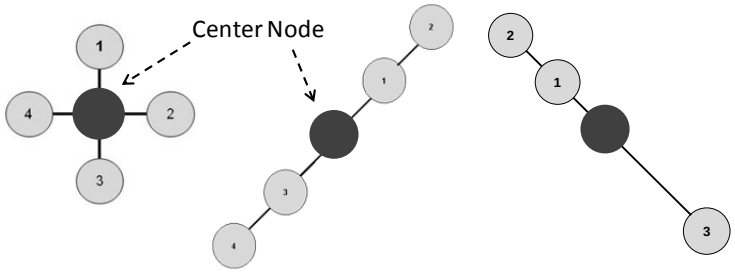


Figure 3.5- Examples of Spatial Templates

By scanning the training image a pattern histogram that records frequency of the observed patterns is constructed. In Figure 3.6, a pattern histogram obtained by scanning the training image using a 4-node template is illustrated. Horizontal axis of the pattern histogram represents data configurations that are observed while scanning whereas the vertical axis reports the frequency of the corresponding pattern (Figure 3.6). Network connectivity is represented through the observed patterns and later implemented in cave/karst pattern simulation. The pattern simulation algorithm is described in Chapter 4.

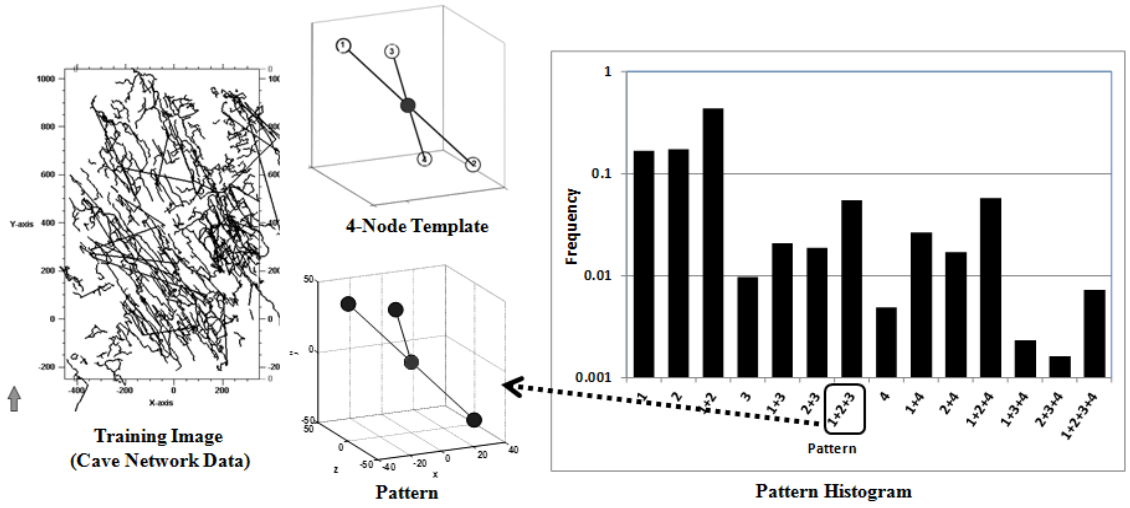


Figure 3.6- Pattern Histogram. Training image is scanned by using the 4-node template and the corresponding pattern histogram is constructed.

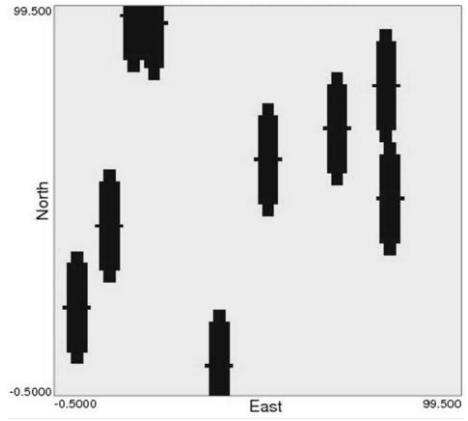
3.5 Non-Gridded MPS Analysis of Synthetic Data Sets

Non-gridded MPS analysis is applied on synthetic data sets in order to test and verify the proposed technique. The synthetic data sets that have prescribed connectivity and orientation are constructed using a Boolean object simulation algorithm, *ellipsim* of GSLIB (Deutsch and Journel, 1998). Preliminary non-gridded MPS analysis and pattern simulation results for various 2D and 3D synthetic data sets are previously given by Erzeybek and Srinivasan (2011) and Erzeybek et al. (2012). This section presents MPS analysis and pattern simulation results obtained for 2D and 3D synthetic fracture networks.

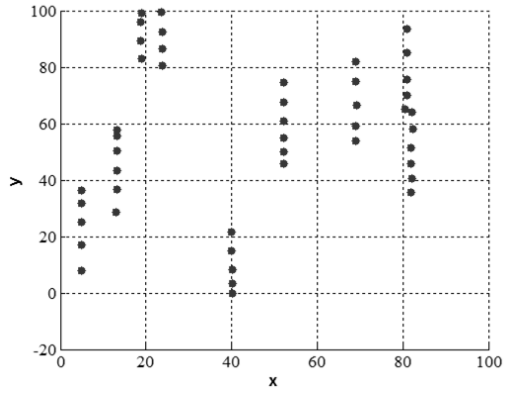
3.5.1 2D Synthetic Network Data

2D synthetic network data is extracted from *ellipsim* models (Deutsch and Journel, 1998). The Boolean simulation algorithm produces objects like ellipses (2D) and ellipsoids based on user-defined parameters such as azimuth and dip angles, object dimension and object density.

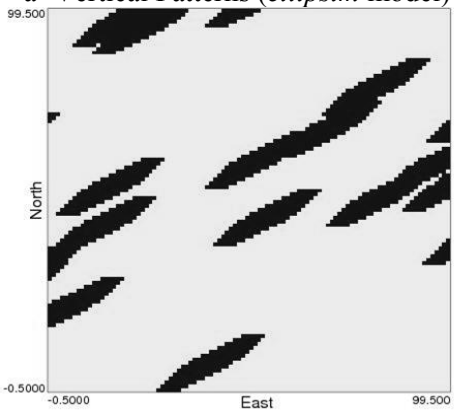
In order to construct 2D synthetic data sets, *ellipsim* is used to create a suite of realizations. Once the ellipsoids representing connected geologic structures are obtained, the central axes of the objects are extracted by digitizing the output model of *ellipsim*. Thus, a set of points following the central axes of the simulated objects is extracted. These points are used as point-set training image and are analogous to the data typically available for a cave network. For demonstration purposes, two different simple ellipsoid models are constructed; vertical and tilted patterns (Figure 3.7) and point sets are digitized from these models. Then, non-gridded MPS analysis and pattern simulation are performed on the synthetic data.



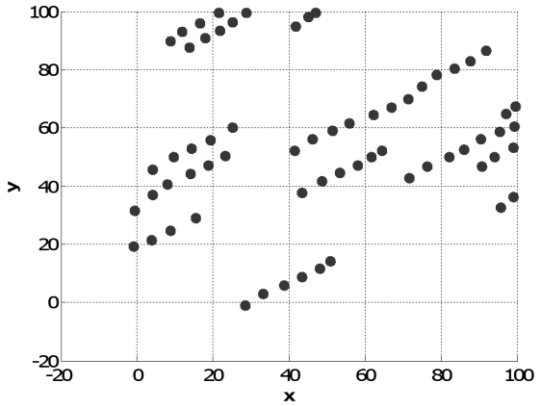
a- Vertical Patterns (*ellipsim* model)



b- Vertical Patterns (Digitized data set)



c- Tilted Patterns (*ellipsim* model)



d- Tilted Patterns (Digitized data set)

Figure 3.7- 2D Synthetic Data Sets

3.5.1.1 Application on Vertical Patterns

The MPS analysis algorithm is applied on the vertical pattern data set (Figure 3.7) and initial screening was performed using five different 2-node templates (Figure 3.8). The corresponding template properties are given in Table 3.1 where the lag tolerance is calculated by multiplying template lag distance by a lag tolerance factor. This tolerance factor yields a tolerance window which has a size at the same scale as the lag distance. The templates are constructed to capture connections at different scales in vertical, horizontal and 30° azimuth directions. Although the pattern orientation is known for this synthetic case, using templates in various orientations demonstrates the capability of the algorithm to capture main trend of the patterns .This step aims to verify the main pattern orientation and identify an optimum template.

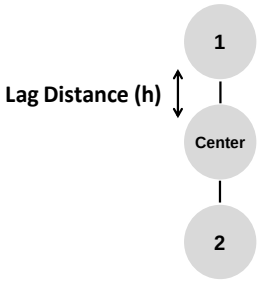


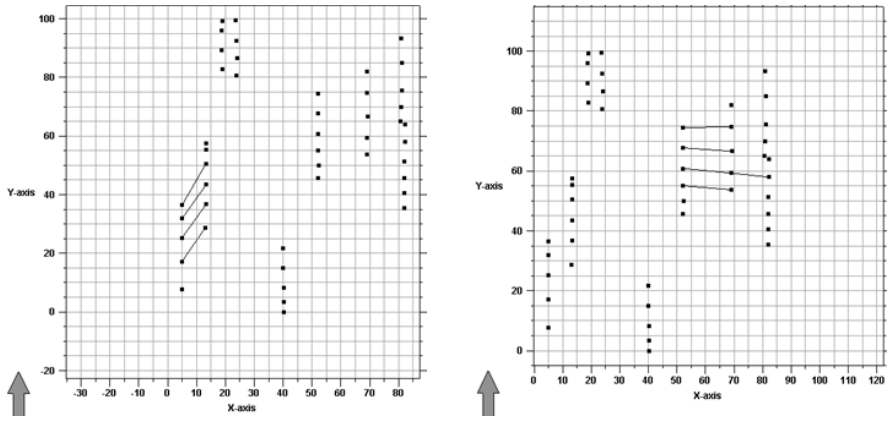
Figure 3.8- 2-Node Template

As evident from Table 3.1, the maximum number of connections is captured by the template in the 0^{o} azimuth (vertical) direction with template nodes that are 5 units apart. The MPS analysis suggests that the connectivity exhibited by the point-set is in the vertical direction and average passage lengths vary from 5 to 15 units. On the other hand, Templates 3 and 4 reported few connections at large lag separation and within a larger tolerance window. This indicates that by specifying large lag and azimuth tolerances, it is possible to pick up spurious patterns and therefore careful examination of the computed pattern statistics is essential.

Table 3.1- Template Properties for Vertical Patterns

Template	Lag Distance (units)	Lag Tolerance Factor	Azimuth Angle (°)	Azimuth Angle Tolerance (°)	Dip Angle (°)	Number of Connections
1	15	1/3	0	15	0	31
2	5	1/3	0	15	0	40
3	15	3/4	30	15	0	8
4	15	1/3	90	15	0	10
5	5	1/3	90	15	0	2

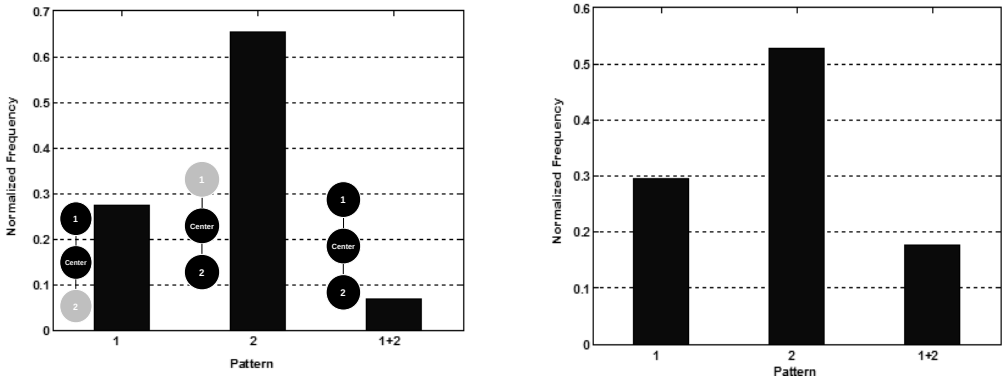
As shown in Figure 3.9, Templates 3 and 4 reported connections that are inconsistent with the original connectivity of the training image. If Template 4 is resized for additional analysis and the point-set training image is scanned by the resized Template 5 only 2 connections are captured. Thus, in order to avoid picking up spurious connections, it is essential to integrate any prior information regarding the training image/network while constructing the templates (for example using the reservoir dip information to orient the template).



a- Template 3 b- Template 4

Figure 3.9- Spurious connections captured by Templates 3 and 4 in vertical patterns. Arrow shows the North Direction.

Templates 1 and 2 and the corresponding pattern histograms are shown in Figure 3.10 and the patterns are illustrated in Figure 3.10a. It is observed that both histograms have similar trends; Pattern 2 has the highest occurrence followed by Pattern 1. On the other hand, Pattern 1+2 has higher frequency in Template 2 histogram indicating that this configuration is captured by using a smaller sized template and continuous pattern length is generally smaller than 30 units (maximum total length of Template 1).



a- Template 1 b- Template 2

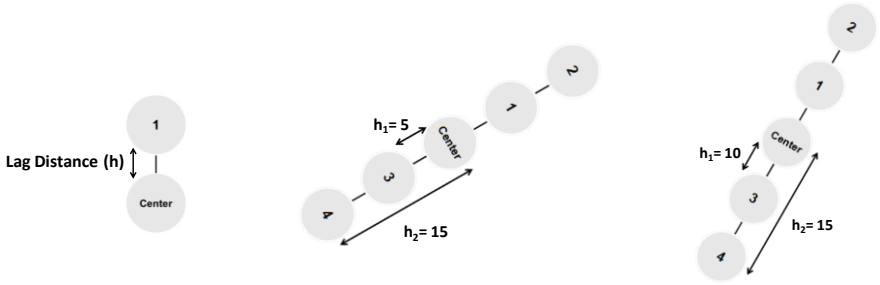
Figure 3.10- Pattern Histograms Corresponding to Templates 1 and 2. Template 1 has a lag distance of 5 units whereas Template 2 has 15 units.

Application of the non-gridded MPS analysis on the simple vertical pattern point set demonstrates that the proposed methodology is capable of capturing MP-statistics of a simple synthetic network. In the subsequent sections, results obtained for more complex point-set training images are presented.

3.5.1.2 Application on Tilted Patterns

MPS analysis of tilted data set is initially performed using 1-node templates (2 nodes including the central node) with different configurations to identify major trends of the patterns in point-set training image. In the preliminary MPS analysis, 6 different 2-

node templates (center + 1-node) are constructed (Figure 3.11a) and their properties are given in Table 3.2. By comparing the number of connections captured by each template, it is observed that the patterns are mostly oriented along 60° azimuth direction.



a- 2-Node Template b- Template 8 c- Template 9

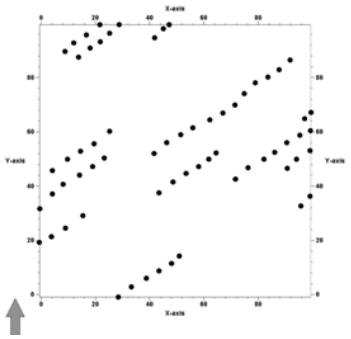
Figure 3.11- Templates used in MPS Analysis for the Synthetic Case with Tilted Patterns.

Table 3.2- Properties of Initial 2-node Templates used to asess the main direction and extent of spatial connectivity of fractures.

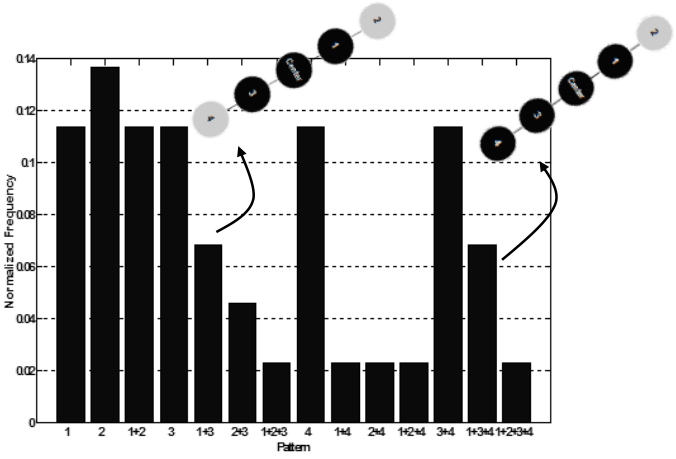
Template	Lag Distance (units)	Lag Tolerance Factor	Azimuth Angle (°)	Azimuth Angle Tolerance (°)	Dip Angle (°)	Number of Connections
1	5	1/3	0	15	0	0
2	15	1/3	0	15	0	4
3	15	1/3	90	15	0	4
4	5	1/3	30	15	0	0
5	15	1/3	30	15	0	5
6	5	1/3	60	15	0	19
7	15	1/3	60	15	0	17

For further MPS analysis, these initial 2-node templates are expanded to 4-node templates both in 30° and 60° azimuth angle directions (Figures 3.11b-c). These additional multi-node templates are assigned to capture continuous patterns along the

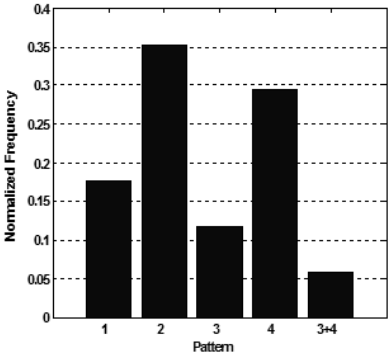
main orientation. Then, MPS analysis of the tilted data set is performed using the additional Templates 8 and 9 (Figures 3.11b-c) and pattern histograms are constructed (Figure 3.12) by using the tolerance values given in Table 3.2. The sum of frequency of all patterns in Template 8 histogram is 74 whereas it is 18 in Template 9 histogram.



Tilted Point Set Training Image



a- Pattern Histogram of Template 8



b- Pattern Histogram of Template 9

Figure 3.12- Pattern Histograms of 4-Node Templates (for Tilted Data Set)

By comparing the pattern histograms in Figure 3.12, it is observed that Template 8 reports a variety of configurations with multiple nodes, whereas Template 9 has patterns mostly limited to simple configurations. This difference in the histograms

indicates that Template 8 is more capable of recognizing the patterns in the training image by reporting several configurations honoring the main features.

MPS analysis of the tilted data set shows that the point-set training image has patterns (patterns with multiple connections such as 1+2, 1+3+4 etc.) that are mostly oriented along 60° azimuth angle. Although the original data set has only one major trend in the 60^{o} direction, Template 9 with 30° azimuth angle reports significant number of connections and patterns that are captured corresponding to point-set data that are close to each other and within the tolerances specified. Since size of the tolerance window affects the captured patterns, assigning tighter lag tolerance might eliminate the spurious patterns reported by Template 9.

3.5.1.3 Valley of Fire State Park Fault Network Modeling

To validate the MPS simulation technique, it is also implemented to model the fault system observed in the Valley of Fire State Park, Southern Nevada. Aerial photograph-based structural interpretation and geology map of the Valley of Fire State Park is previously reported (Flodin and Aydin, 2004) (Figure 3.13). Liu and Srinivasan (2004) used the fault network data for a portion of the park as a training image and used the resultant statistics to demonstrate the application of a conventional grid-based MPS algorithm for fracture network modeling (Figure 3.14).

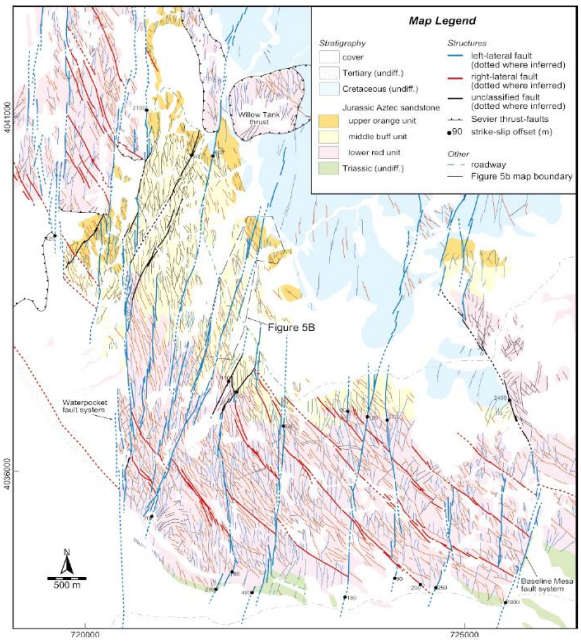


Figure 3.13- Aerial Photograph-Based Structural Interpretation and Geology Map (Flodin and Aydin, 2004)

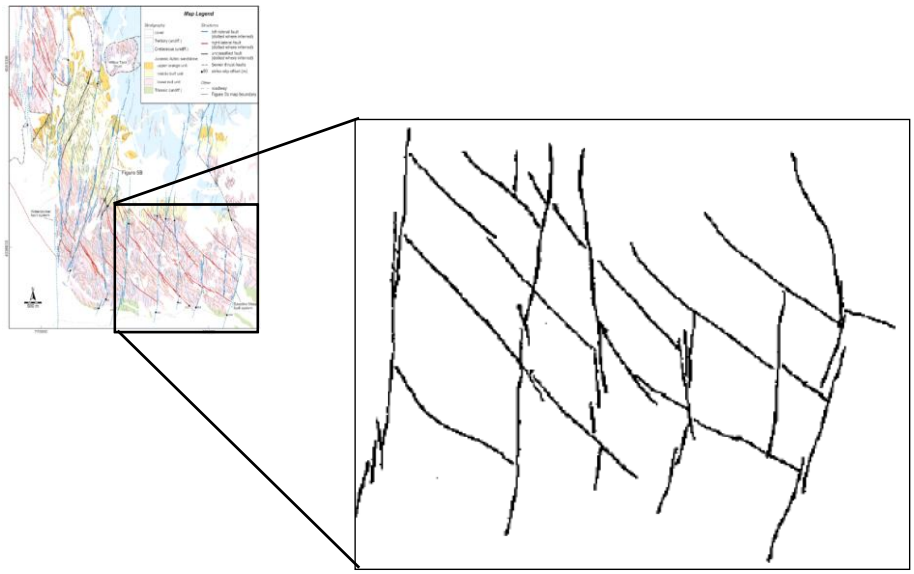


Figure 3.14- Training Image for Fault System used in Liu and Srinivasan (2004)

In order to utilize the Valley of Fire fault data for the current work, the analog model used by Liu and Srinivasan (2004) is digitized to obtain a point-set training image representing the fault network (Figure 3.15).

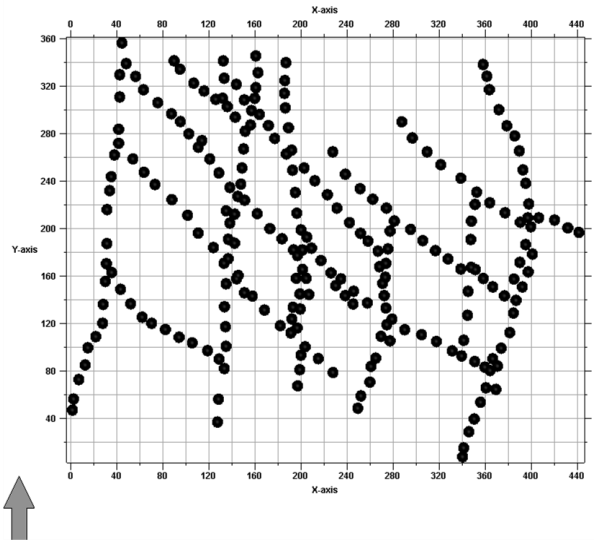


Figure 3.15- Digitized Analog Model Used as Training Data for the Grid-less MPS algorithm. Arrow shows the North Direction.

The digitized point-set data is first scanned using 19-node generic templates (Figure 3.16) to identify an optimum template size and orientation. Lag tolerance factor of 1/3 and azimuth angle tolerance of 10° are assigned for the tolerance window. Properties of the more significant generic templates and number of connections captured are given in Table 3.3.

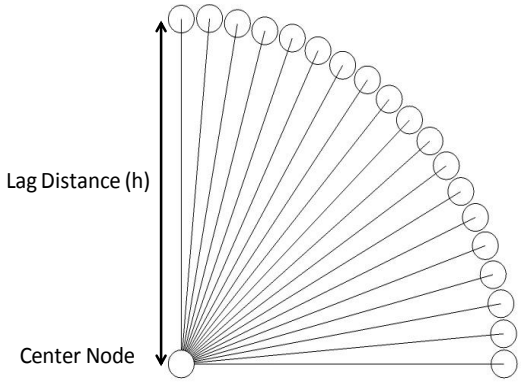
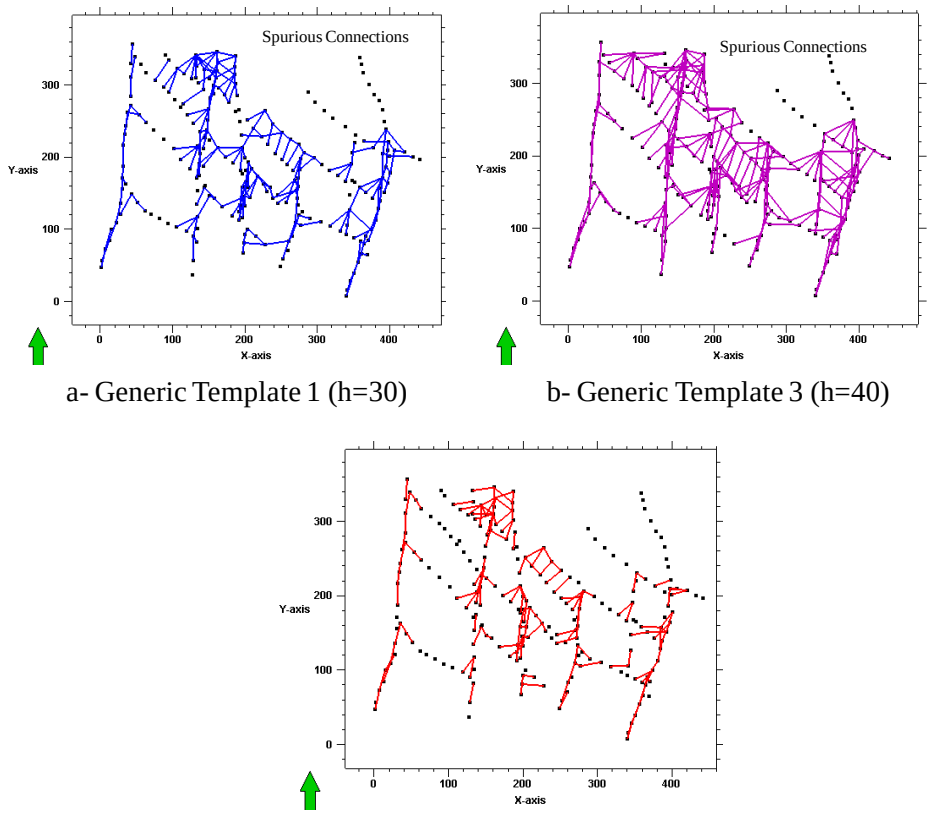


Figure 3.16- 19-Node Generic Templates used for the Valley of Fire Fault Data

Table 3.3- Properties and Statistics for the significant Generic Templates Applied on the Valley of Fire Fault Network

Template	Lag Distance h (units)	Lag Tolerance Factor	# of Connections Captured
1	30	1/3	226
2	20	1/3	97
3	40	1/3	371
4	10	1/3	34
5	25	1/3	148

It is observed that the templates with the larger lag separation capture more connections compared to the smaller ones. Since generic template analysis provides an overview of the network, they are used to guide the construction of more complex spatial templates for further analysis. The connections captured by Templates 1, 3 and 5 are compared and it is determined that the large-sized templates start capturing spurious connections that do not accurately representing network connectivity (Figure 3.17).



a- Generic Template 1 (h=30)

b- Generic Template 3 (h=40)

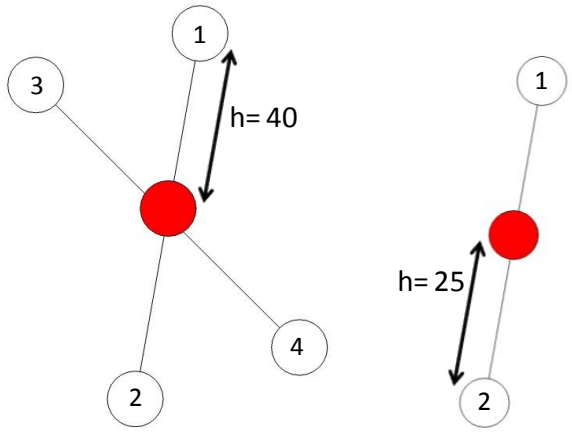
c- Generic Template 5 (h=25)

Figure 3.17- Spurious Connections Captured by some of the Large-sized Generic Templates Applied on the Valley of Fire Fault Network.

Compared to Templates 1 and 3, Template 5 has less number of spurious connections that are mostly observed at faults intersections and areas at which the faults are closer to each other. Despite the minor artifact connections, it is concluded that Template 5 with a lag distance of 25 units between template nodes is the most reliable for capturing small-scale connections. On the other hand, large scale continuous patterns are captured by Template 3

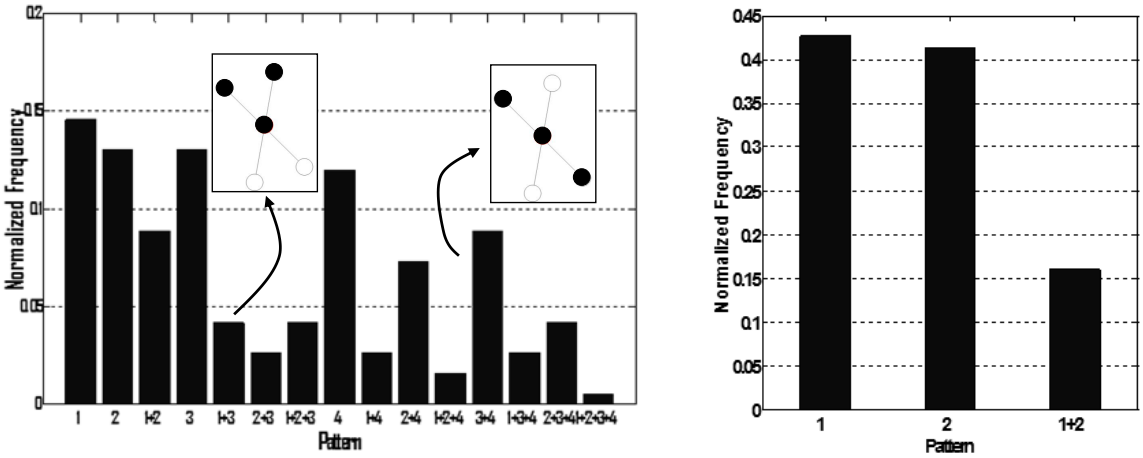
Two different spatial templates are constructed for MPS analysis (Figure 3.18). The first template is used for capturing large scale features along the major trend of 10° and -45° azimuth angle directions. The second template is designed for small scale

features oriented in 10° azimuth direction. Tolerance window is defined by an azimuth tolerance of 15° and a lag tolerance factor of 1/3 for both templates. By scanning the training image (Figure 3.15) using these spatial templates, pattern histograms at multiple scales are calculated (Figure 3.19).



a- Template 1 b- Template 2

Figure 3.18- Spatial Templates used for computing the pattern statistics of the Valley of Fire Fault Network.



a- Template 1 b- Template 2

Figure 3.19- Pattern Histograms for the Valley of Fire Fault Network corresponding to the template at two scales.

3.5.2 3D Synthetic Fracture Network

In order to further test and verify the MPS algorithm, a 3D synthetic fracture network is developed (Figure 3.20). The synthetic data set is constructed by modifying the program *ellipsim* in GSLIB (Deutsch and Journel, 1998) for simulation of objects in a reservoir. Initially, an object simulation of three different types of ellipsoids is performed using the modified *ellipsim* program and the object center locations are stored with corresponding feature type. The features are in 20°, 35° and 50° degrees azimuth angle direction with 30°, 10° and 20° degrees dip angles respectively. Once the center trajectory of the ellipsoids is obtained, points along the corresponding feature orientation are sampled at an irregular spacing. A minimum distance separation of 0.5 units between the sampled points is specified. This is to mimic the irregularly spaced point-survey data that might be available in the case of a real cave.

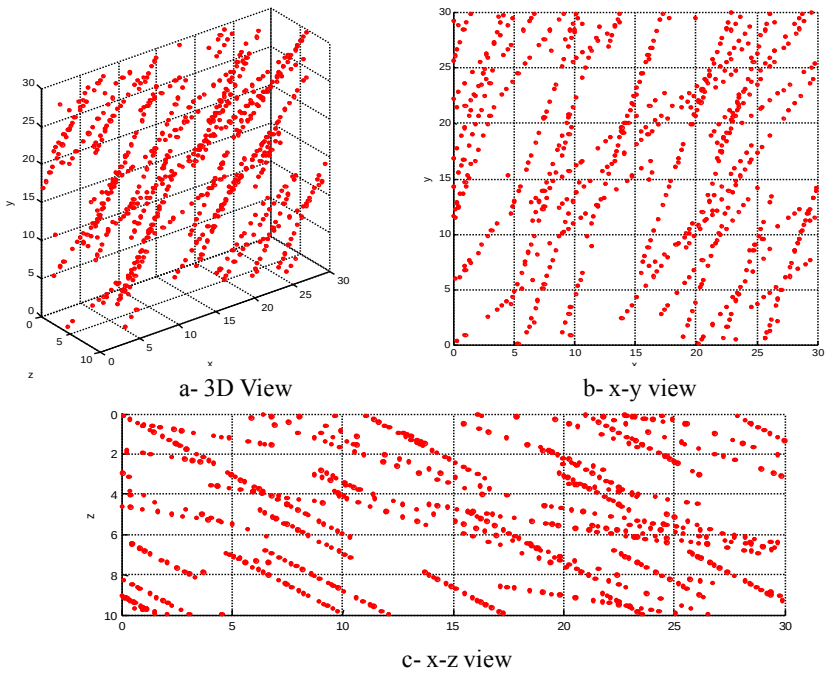


Figure 3.20- 3D synthetic fracture network obtained by randomly sampling points along the center-line of ellipsoids.

MPS analyses of 3D synthetic network are initially performed using the 19-node generic templates. These generic templates are used to determine azimuth and dip orientation of the main features. Non-gridded MPS analysis is performed using a lag distance of 1 units, lag tolerance factor of 1/3, azimuth tolerance of 6° and dip angle tolerance of 10°. The nodes are placed at a succession of azimuth angles spanning a 90° range with 5° increments. First, dip angle of the generic template is systematically varied from 0° to 60° and the number of fracture connections captured by each generic template are compared (Figure 3.21).

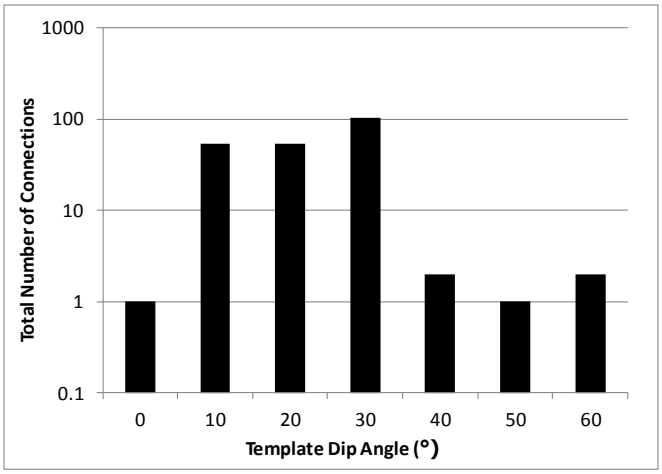


Figure 3.21- Comparison of Total Number of Connections Captured by the entire set of nodes in Generic Templates (3D Synthetic Fracture Network)

Significant number of connections are captured by templates with 10°, 20° and 30° dip angles (Figure 3.21) and this observation is consistent with the original 3D synthetic network data which has features following these dip trends. Also for each generic template, the azimuth angle along which the maximum number of connections recorded is determined. By using the generic templates, an initial and quick assessment of the feature orientations is performed and the orientations of main features are determined.

Once the overall dip and azimuth orientations of the connected features have been established, pattern inference procedure is performed using 4-node spatial templates along the major feature orientations (Figure 3.22) and the spatial template properties are given in Table 3.4. 4-node templates are selected for the analysis to capture continuous patterns and restricting the number of nodes eliminates the chance of capturing spurious connections. Templates with additional nodes can also be applied for uncertainty assessment purposes. Using these 4-node spatial templates, the 3D synthetic fracture network data is scanned and the MP-statistics of the synthetic fracture network is calculated (Figure 3.23). Computed pattern histograms are implemented in network simulation.

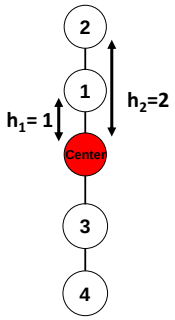


Figure 3.22- 4-Node Spatial Template for MPS Analysis

Table 3.4- Properties of the complex spatial template used for inferring the spatial characteristics of the 3D Synthetic Fracture Network

Template	Lag Tolerance Factor	Azimuth Angle (°)	Dip Angle (°)
1	1/2	20	30
2	1/2	35	10
3	1/2	50	20

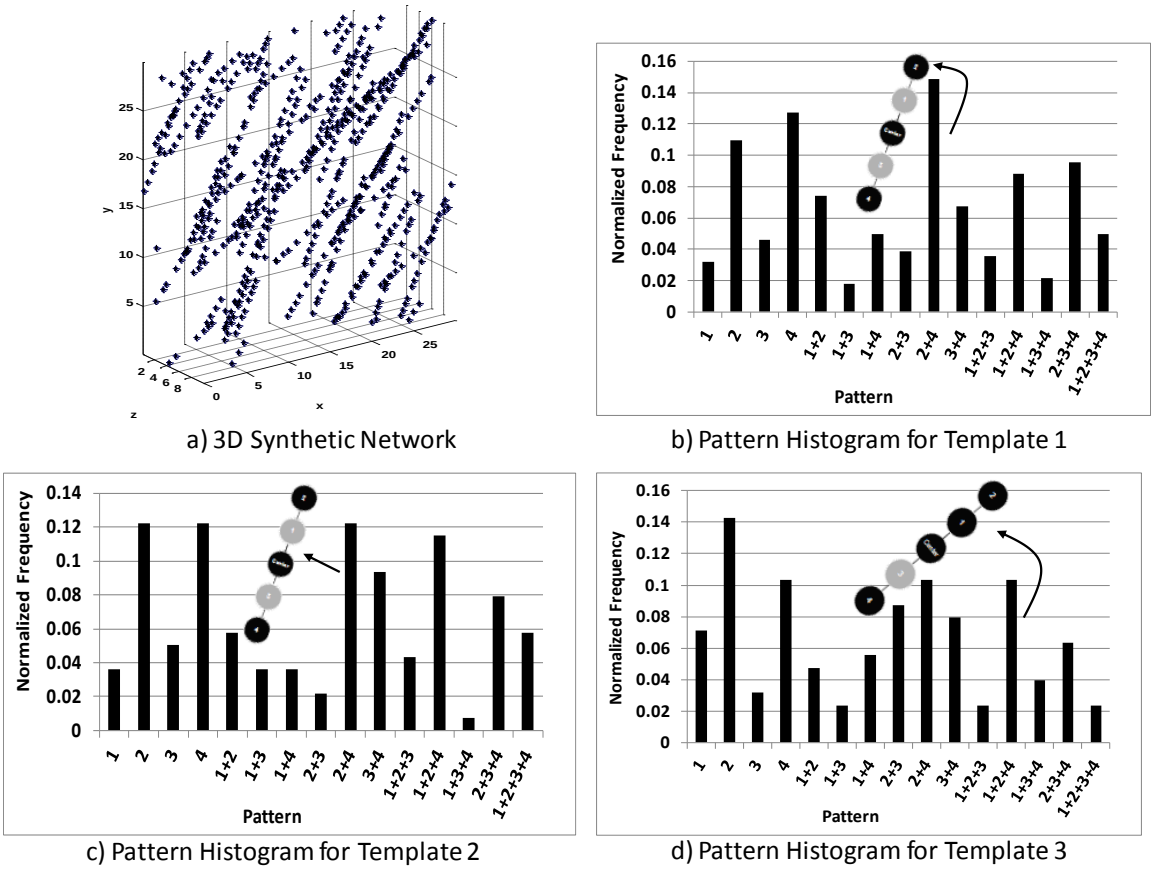


Figure 3.23- Pattern Histograms of 3D Synthetic Fracture Network

Pattern histograms shown in Figure 3.23 have a variety of patterns with single and multi-node configurations. Since Template 1, 2 and 3 have different orientations (Table 3.4), each histogram reports patterns along the corresponding template configuration. It is observed that templates capture continuous features (i.e. configurations with multiple nodes such as patterns 2+4, 1+2+4 and 2+3+4). Since continuous patterns are reported by all of the templates, Templates 1, 2 and 3 are successful at capturing patterns along their corresponding orientations.

It can be concluded based on these synthetic examples that the non-gridded MPS analysis is successful at characterizing networks with complex structures and multiple

orientations. After inferring the MP-histograms, pattern simulation of the point-set data is performed by using the spatial templates. In Chapter 4, pattern simulation algorithm developed for modeling cave/fracture networks is explained and pattern simulation results of the synthetic data sets are presented.

Chapter 4: Pattern Simulation Algorithm

The MP-statistics inferred from the training image are used to constrain the pattern simulation and to represent connected geologic features. The simulation is conditioned to sparse data in the form of spatial locations with cave facies or coordinates of cave structures. In the case of very large data sets and limited computational resources, a subset may be retained by selecting the conditioning set at random. In order to avoid clustering of selected data points, a minimum separation distance may be assigned between the conditioning locations. The pattern simulation is performed using a pattern growth-based algorithm; simulated network starts growing at the conditioning data locations that are visited randomly.

The pattern simulation workflow is summarized in Figure 4.1. At the beginning of the simulation, pattern with the highest frequency in the MP-histogram is applied at conditioning data locations. Nodes in the vicinity of conditioning data locations such that the spatial template centered on those nodes has at least one conditioning data on any of the template-nodes are deemed "simulatable" nodes. Then, simulation proceeds sequentially by randomly selecting one of the "simulatable nodes" as the next "simulation node". As the simulation proceeds, the "simulatable node" list continuously evolves.

Once the "simulation node" is selected, data configuration around the simulation node is scanned using the same non-gridded template that was used for computing the mp histogram. Pre-defined lag and angle tolerances are employed while matching the simulation data configuration with the pattern statistics for the template. At this step, the algorithm has a user-defined option to use a single template or a suite of templates. The current data configuration around the simulation node template is searched among all the templates and the one that matches with the highest probability is retained.

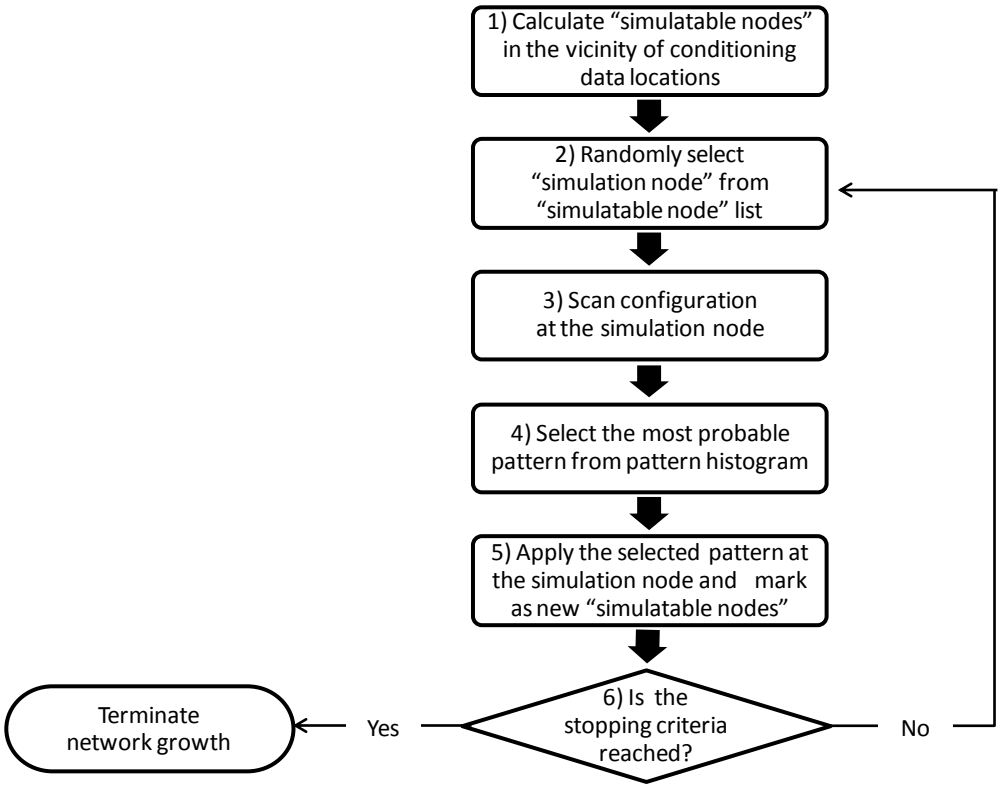


Figure 4.1 - Pattern Simulation Algorithm Steps

Based on a Monte Carlo draw, the simulation pattern is extracted for the remaining nodes of the simulation template. A location is randomly selected within the tolerance window centered at the node corresponding to the simulated pattern and is assigned to be either a cave node or a non-cave node. This is repeated at all the other nodes corresponding to the simulated pattern. By randomly placing the new nodes inside the tolerance window instead of assigning the simulated pattern at exactly the center node locations, tortuous nature of cave/fracture networks is accurately reproduced.

Once the new data locations are marked, they are included in the "simulatable node" list. All the new data locations are considered as "simulatable node" regardless of whether they are marked as cave or non-cave. Thus, the algorithm enables the network to

grow from non-cave/ non-fracture locations as well as terminate network growth. Since in modern caves, the known extent of the cave network is controlled by the accessibility of the cave passages, it is possible to have additional caves and network structures that are unknown and remain inaccessible but may lead to other sections of the network that may be accessible. By considering non-cave/ non-fracture locations as "simulatable nodes", the pattern simulation algorithm accounts for the location of unknown and potential cave/fracture structures.

The simulation procedure continues until the desired stopping criterion is reached. A user-defined stopping criterion can be assigned on the basis of maximum cave/fracture density, maximum total network length, maximum deviation from original pattern histogram or maximum simulation iteration steps for practical purposes.

The pattern simulation algorithm has a variety of options that are useful for field scale application; use of single or multiple templates, pattern simulation with or without histogram filtering, simulation with quadrant analysis, simulation with surface dip map and concurrent cave thickness simulation options. Description of these simulation options are presented in the following sections.

4.1 Pattern Simulation Using Multiple Templates

The spatial distribution of cave/fracture network is affected by various factors and in some cases, network passages might have several major trends. This supports the use of multiple spatial templates with different orientations to model complex network structures. The pattern simulation algorithm is capable of handling multiple major orientations by integrating multiple templates and corresponding MP-histograms.

The multi-template methodology is similar to the single template case with minor differences. Prior to Step 1 (Figure 4.1), patterns in the entire suite of MP-histograms are examined and the most probable configuration among them is determined. Then, the

selected pattern is applied in Step 1 to calculate "simulatable nodes" in the vicinity of the conditioning data. The simulation procedure follows the workflow in Figure 4.1 and at Step 3, data configuration at the simulation node is scanned by using the entire set of templates. Then in Step 4, MP-histograms of all the templates are compared and the pattern to be applied is selected that has the highest probability.

The use of multiple templates enables one to handle complex networks with more than one major trend. This implementation yields accurate reproduction of the network while integrating different trends with features at similar scales.

4.2 PATTERN SIMULATION WITH HISTOGRAM FILTERING

In order to improve simulation results, histogram filtering is also applied. In histogram filtering, at a certain simulation step, the simulated patterns with frequency below a user defined threshold are eliminated. The frequency of histogram filtering is controlled by a user defined histogram control step number. At every histogram control step, pattern histogram of the simulated network is constructed and, the data locations with pattern frequency below a threshold are eliminated. This procedure serves to reduce the noise in the simulation result by filtering the simulated nodes which yield patterns that are not physically observable in the target network.

Histogram filtering is a practical tool to control network growth in case there is limited information on the spatial extent of the target cave network. Elimination of low frequency patterns results in simulated patterns that are mostly observable in the training image. By reducing the noise in simulation results, the target MP-histogram is successfully reproduced and more accurate network simulation results are obtained.

4.3 QUADRANT ANALYSIS IN PATTERN SIMULATION

Pattern simulation algorithm allows network growth from both cave and non-cave marked data locations and random selection of simulation nodes might result in clustering of simulated passages that is inconsistent with the target network or the regional geology. In order to avoid this drawback, a quadrant analysis option is included in the pattern simulation.

Quadrant analysis also serves as a simulation control mechanism similar to histogram filtering. Prior to the simulation, quadrant analysis is performed on the training image or any prior data to determine the spatial extent of the target network. Since the purpose of the quadrant analysis is to broadly identify spatial distribution of the passages in the training image or in the target network, quadrant size is assigned large enough to capture clusters of network structures. In cave modeling, quadrant analysis is performed on cave network maps and a histogram that reports the number of nodes in a quadrant is constructed. When modeling a paleokarst reservoir, quadrant analysis is conducted using well information. Wells with identified paleocave facies are considered as nodes along the paleokarst network and quadrant frequency is determined based on the number of wells with paleocave facies in that particular quadrant. Prior quadrant information (i.e. quadrant size and number of quadrants in horizontal and vertical) and corresponding cluster histogram are used in the pattern simulation to define homogeneous simulation domains.

The simulation workflow is not affected by the quadrant analysis except for a minor variation while updating “simulatable node” list. At the quadrant check step, quadrant histogram of the entire “simulatable node” list is constructed based on the given quadrant properties. Then, frequency of each quadrant is compared to the prior information. Data locations in the quadrants exceeding prior quadrant frequencies are

eliminated from the "simulatable node" list. This elimination results in termination of network growth in quadrants with scarce clustering of passages. Similar to the histogram filtering, quadrant analysis is performed at every "quadrant check step" that is also user defined.

4.4 Integration of Surface Dip Maps in Pattern Simulation

Statistics inferred from the training images are implemented in network modeling. Thus, the templates used in network simulation are constructed along the major dip directions of the training images. On the other hand, it is challenging to identify main orientation of the paleokarst network in subsurface using only well information. Therefore, additional geological knowledge should be included for re-orientation of templates in pattern simulation. In the case of epigenic paleokarst networks, it is adequate to integrate a dip map of the layer hosting the paleokarst. Since network is conformable, current orientation of the paleokarst features will follow the host formation. Therefore, integration of host formation dip map is a practical tool for re-orientation of the templates. An application of this procedure is demonstrated in this dissertation. For hypogenic networks, pre-karst structural geology as well as the host formation geology might be considered for template re-orientation.

Surface dip map integration does not affect the pattern simulation workflow except a minor modification that is applied whenever the template is used. First, the closest node to the simulation location is determined on the surface dip map. Then, template is oriented along the dip direction at that particular location. This same dip direction is used i) for the initialization step to calculate "simulatable nodes", ii) while scanning configuration at the simulation node, and iii) while calculating new data locations at Step 5.

Although the integration of surface dip maps is not an essential requirement for modeling cave/fracture network, it is a practical tool for accurate simulation of the subsurface features by honoring the geology. Moreover, the procedure allows modeling of networks that are residing in formations with varying dip such as ramps and anticline structures.

Pattern simulation algorithm and aforementioned simulation options are applied to various data sets; 2D and 3D synthetic fracture networks, 2D fault network system in Valley of Fire State Park of Southern Nevada, 3D model of Wind Cave located in South Dakota and Yates Field of West Texas which has an epigenic type of paleokarst network in the main producing San Andres Formation. The pattern simulation results for the 2D and 3D synthetic cases are given in the following sections of this chapter.

4.5 PATTERN SIMULATION OF SYNTHETIC DATA SETS

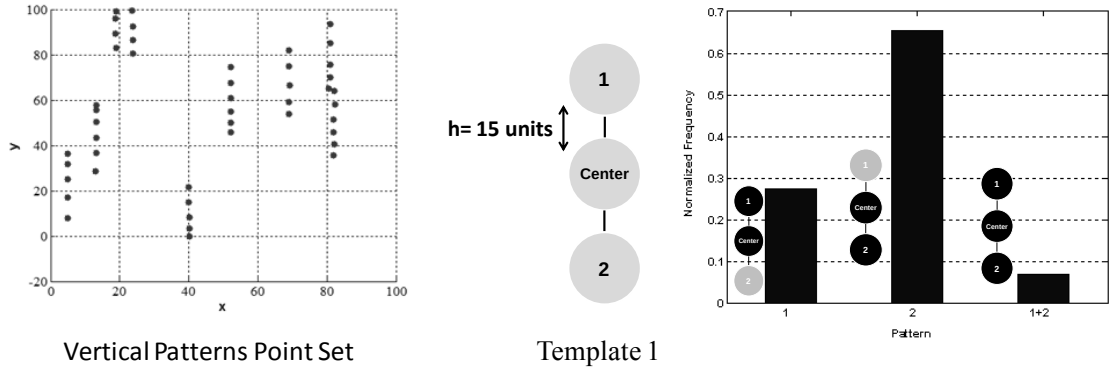
4.5.1 2D Synthetic Network Data

Pattern simulation of the 2D synthetic point-sets is performed by using the inferred MP-statistics given in Chapter 3. Several realizations are obtained by implementing various pattern simulation options.

4.5.1.1 Simulation of Vertical Patterns

In simulation of vertical patterns (Figure 4.2), two different templates and corresponding MP-histograms are used. First set of realizations are obtained by using only Template 1 (Figure 4.2) and the simulation results are shown in Figure 4.3. The conditioning data for the simulation is obtained by randomly sampling 15 conditioning locations from the synthetic point-set maintaining a minimum separation distance of 10 units. Pattern growth was terminated at 100 steps. An azimuth angle tolerance of 10° and lag distance tolerance factor of 2 is specified. The size of the tolerance window is slightly

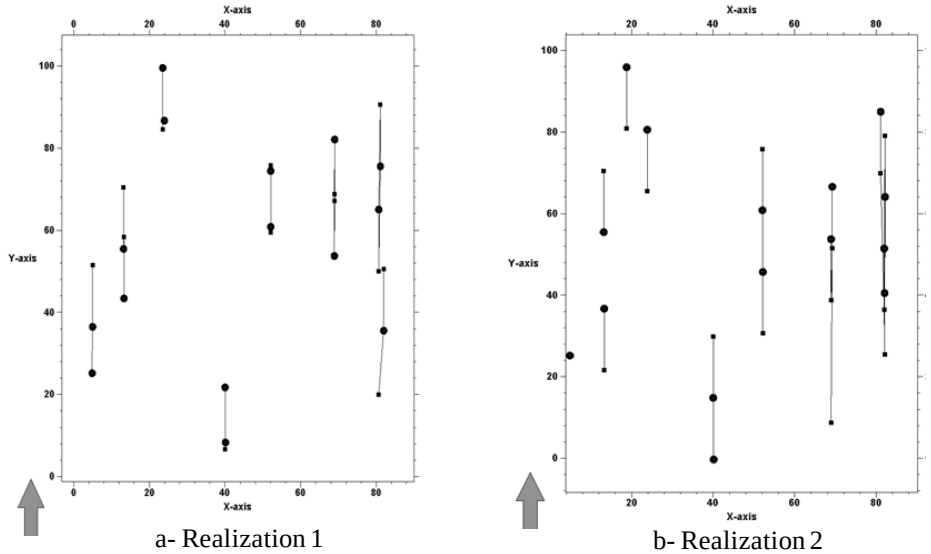
large compared to the one used for inferring the pattern statistics. This was done mainly to be able to determine the pattern in the nodes surrounding a simulation node.



Vertical Patterns Point Set Template 1

Figure 4.2- Template 1 and Pattern Histogram used in Simulation of Vertical Patterns

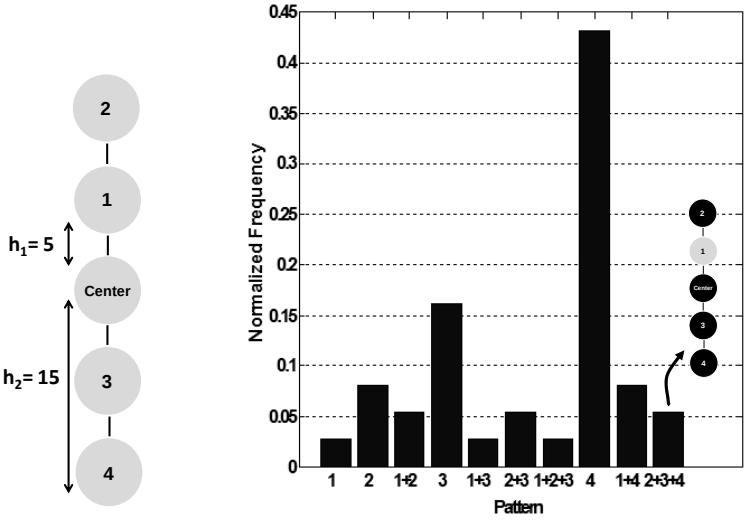
Compared to the original image (Figure 4.2), the pattern simulation yields reasonable results by reproducing vertical patterns (Figure 4.3). The feature orientation and average length are well reproduced as indicated by the pattern histogram in Template 1.



a- Realization 1 b- Realization 2

Figure 4.3- Results Obtained by Template 1 (Circles: Conditioning Data; Lines and Squares: Simulated Connections and Points. Arrow shows the North direction.)

Another set of realizations are obtained using a template that is a combination of Templates 1 and 2 given in Table 3.1. Since Template 2 captured the highest number of connections as seen in Table 3.1, it is used in the construction of the combined template that will help reproduce small-scale structures observed in the vertical point-set. The combined template and its pattern histogram constructed in Chapter 3, are implemented in the pattern simulation and they are shown in Figure 4.4. Also, Pattern 2+3+4 is illustrated in the pattern histogram as an example. It is observed that small patterns usually have higher number of occurrence whereas configurations with multiple nodes have lower frequency indicating that the continuous patterns are limited.

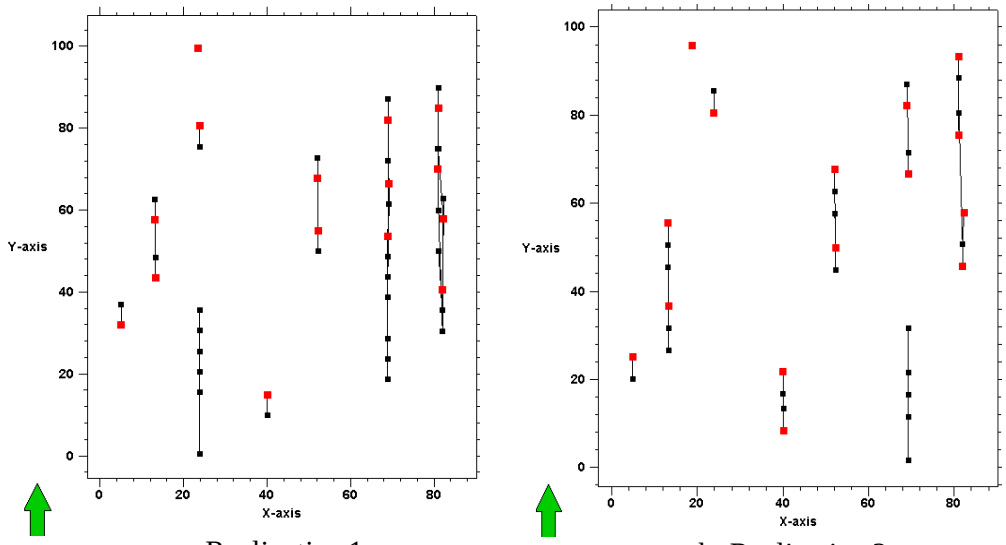


a- Combined Template b- Pattern Histogram

Figure 4.4- Combined Template and Pattern Histogram

Two of the realizations obtained using the combined template are shown in Figure 4.5. The number of conditioning data, minimum separation distance of conditioning locations and the size of the tolerance window are kept the same as in the previous simulation with Template 1.

It is observed that by implementing the combined template (Figure 4.4), pattern simulation results are improved; vertical patterns are successfully reproduced and the feature lengths are more consistent with the original point-set training image (Figure 4.2). The patterns start growing from the conditioning data and if the simulated node is outside of the target simulation area, it is eliminated.



a- Realization 1 b- Realization 2

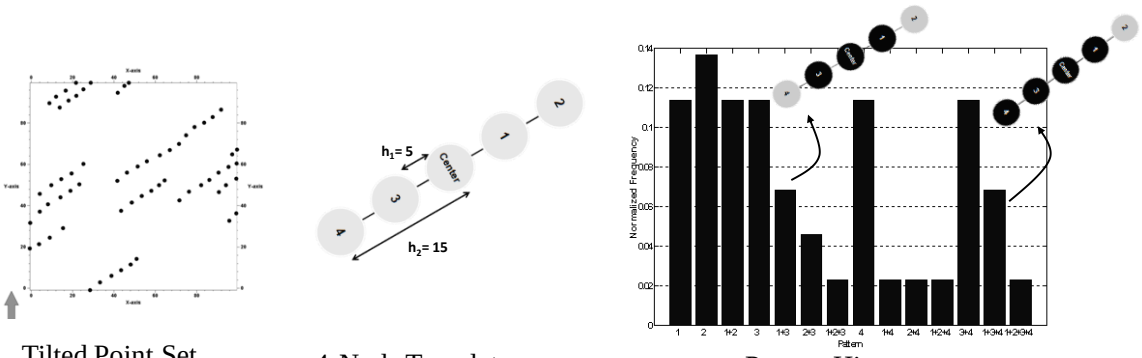
Figure 4.5- Realizations for Vertical Patterns Obtained using a Combined Template (Red Square: Conditioning Data; Black Lines and Squares: Simulated Connections and Points. Arrow shows the North direction.)

Application of the non-gridded MPS analysis and pattern simulation algorithm on the simple vertical pattern point set demonstrates that the proposed methodology is capable of reproducing the simplistic patterns observed in training set with vertical patterns. In the subsequent sections, results obtained for more complex point-set training images are presented.

4.5.1.2 Simulation of Tilted Patterns

Once the MP-statistics are calculated for the tilted point-set, pattern simulation is performed. The first set of realizations is obtained using only a 4-node template and

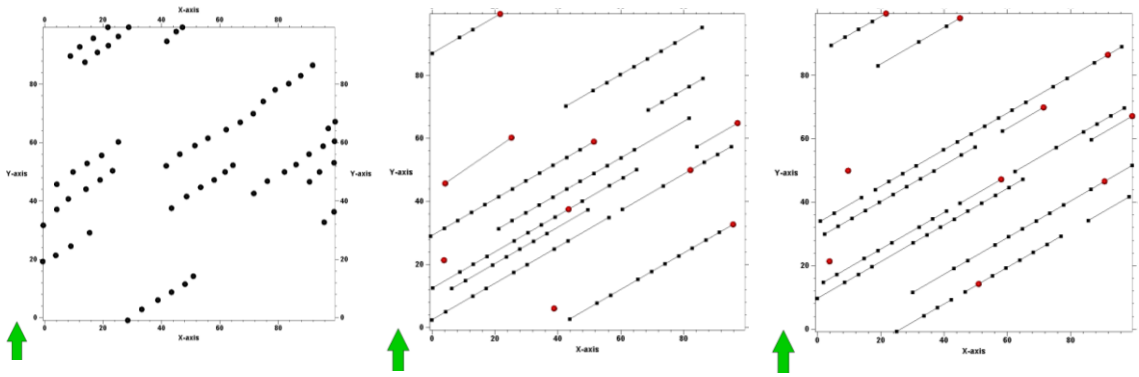
corresponding pattern histogram (Figure 4.6). Tolerance window defined for MPS analysis is used for the pattern simulation (Table 3.2).



Tilted Point Set Training Image | 4-Node Template | Pattern Histogram

Figure 4.6- 4-Node Template and Pattern Histogram used for Simulation of Tilted Patterns

Two such realizations are shown in Figure 4.7 and they are obtained by using 10 conditioning data points randomly sampled from the point-set training image.



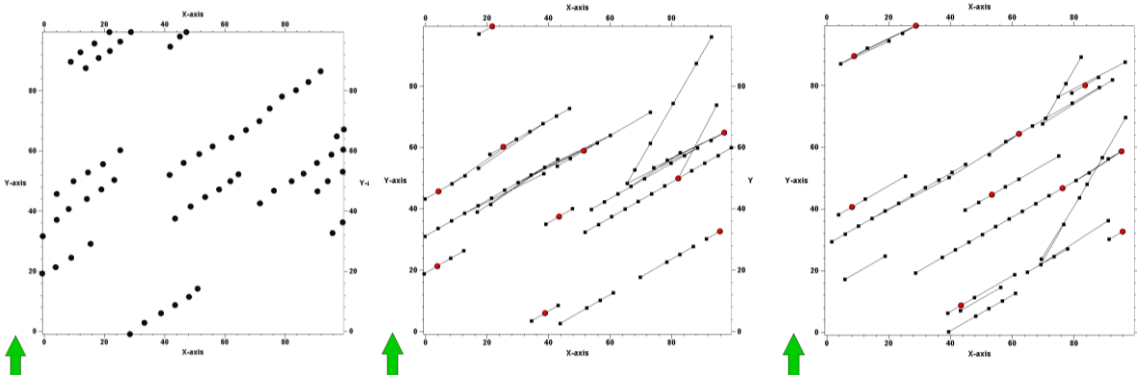
Tilted Point Set Training Image | a- Realization 1 | b- Realization 2

Figure 4.7- Realizations Obtained Using a Single 4-node Template (Red Points: Conditioning Data; Black Lines and Squares: Simulated Connections and Points. Arrow shows the North direction.)

The pattern simulation algorithm successfully reproduces the patterns observed in the original training image. By using only a 4-node template which has nodes along the

major orientation of the training patterns, simulated networks of connected and elongated features are obtained. Patterns follow the main trend of 60° azimuth angle. Although large sized structures are dominant in the realizations, small sized features are also reproduced.

To understand the effect of using multiple templates on the simulated images, a second set of realizations are obtained by using applying three templates; two 4-node templates following 60° and 30° azimuth angle respectively and a 2-node template with nodes oriented in the vertical direction with a lag distance of 15 units (these templates are given in Figure 3.11 and pattern histograms are shown in Figure 3.12). These realizations are obtained by concurrently using the 3 templates in the simulation and the results are shown in Figure 4.8. Although vertical template has a low frequency of occurrence as shown previously (Table 3.2), it is implemented for sensitivity purposes to observe the effect of a template that is not aligned with the main direction of connectivity. For better comparison to the single template results, the same number of conditioning data and same tolerance window size are used.



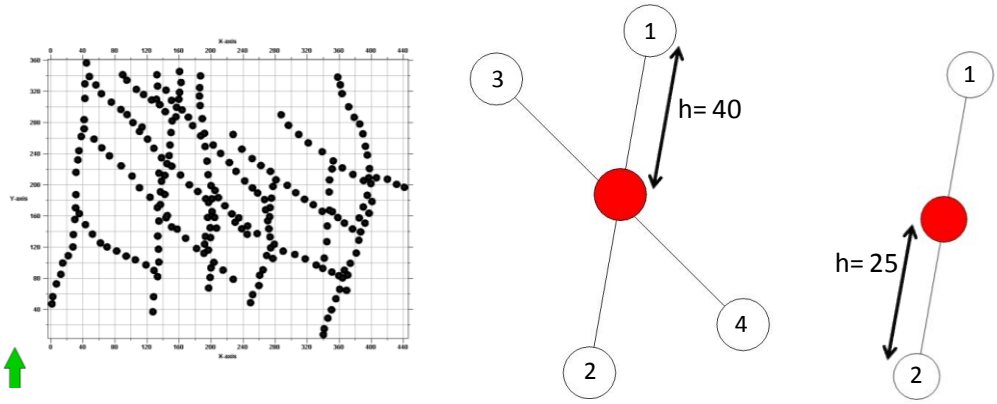
Tilted Point Set Training Image a- Realization 3 b- Realization 4

Figure 4.8- Realizations obtained using Multiple Templates (Red Points: Conditioning Data; Black Lines and Squares: Simulated Connections and Points)

Pattern simulation with multiple templates results in simulated networks with large and small sized structures in different orientations. It is observed that the simulated features are in both 30° and 60° azimuth angle. Since the vertical 2-node template has a low frequency of occurrence (Table 3.2), no vertical features are observed in the simulation. This suggests that the pattern histograms filter the effect of spurious templates that might be specified during the simulation process.

4.5.1.3 Fault Network Modeling Valley of Fire State Park

Two different spatial templates are constructed for MPS analysis and pattern simulation of the Valley of Fire State Park fault network (Figure 4.9a-c). The first template is used for capturing large scale features along the major trend of 10° and -45° azimuth angle directions. The second template is designed for small scale features oriented in 10° azimuth direction. Tolerance window is defined by an azimuth tolerance of 15° and a lag tolerance factor of 1/3 for both templates. By scanning the training image (Figure 4.9a) using these spatial templates, pattern histograms at multiple scales are calculated (Figure 4.10).

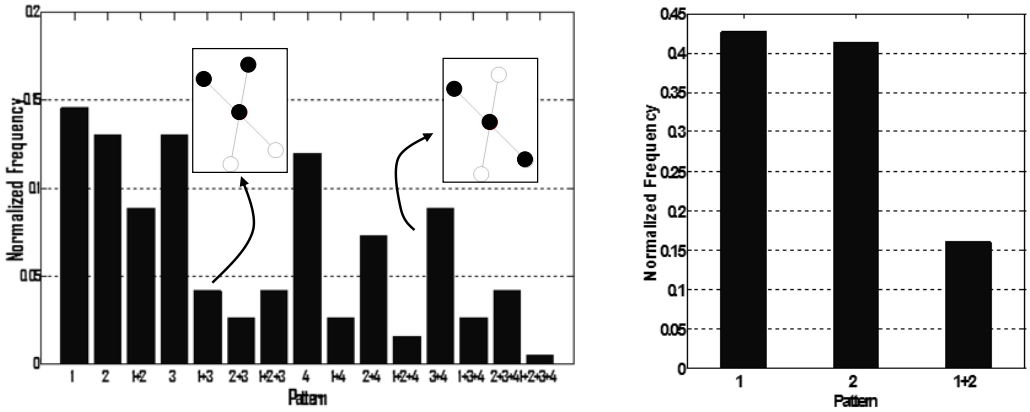


a- Digitized Analog Model of Valley of Fire Fault Network

b- Template 1

c- Template 2

Figure 4.9- Spatial Templates used for computing the pattern statistics of the Valley of Fire Fault Network.

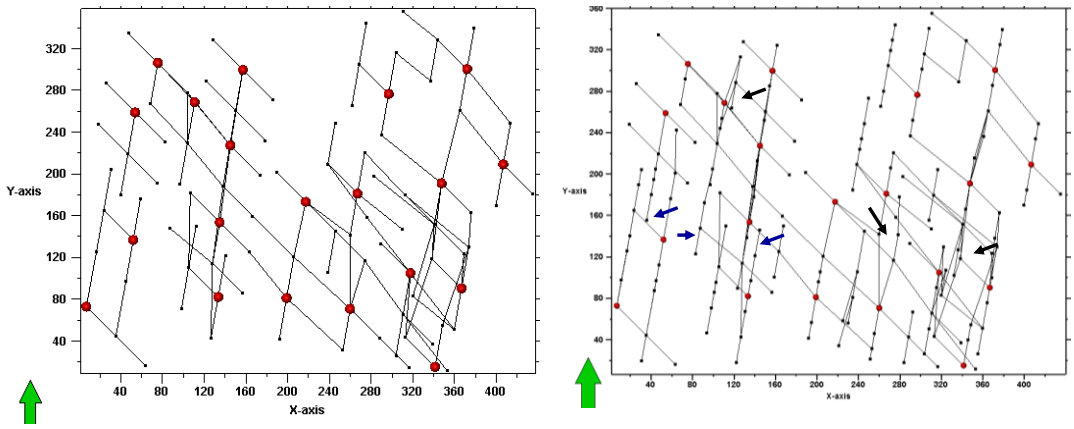


a- Template 1 b- Template 2

Figure 4.10- Pattern Histograms for the Valley of Fire Fault Network corresponding to the template at two scales.

After inferring the MP-histograms, pattern simulation of Valley of Fire fault network is performed by implementing the templates in Figure 4.10. The network is simulated following a multi-grid approach; initial large scale fault simulation using Template 1 (4.9b) and 204 nodes are simulated in this step. Then, consequent small scale features are simulated by Template 2 (4.9c) and 200 additional nodes are obtained (Figure 4.11).

The conditioning data used for simulating the fault network is obtained by randomly sampling 20 locations from the point-set fault data. The pattern growth algorithm described in the previous chapter is applied and the simulation is terminated at 100 iteration steps. The simulated network at the end of the coarse scale simulation is presented in Figure 4.11a. The large-scale connections of the fault network are successfully reproduced.



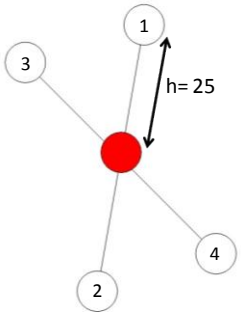
a- Fault Network after Step 1 b- Fault Network after Step 2

Figure 4.11- Multiple-Point Simulation Result for Valley of Fire Fault Network (Red Points: Conditioning Data; Black Points and Lines: Simulated Nodes and Connections. Arrow shows the North direction.). The simulation was first performed using a coarse scale template (left) and subsequently using a small-scale template (right). Black arrows show the curvilinear features whereas blue arrows show simulated intersecting faults.

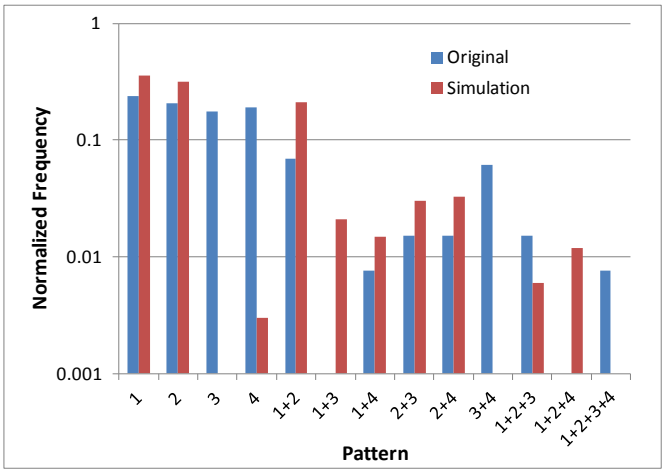
Next, the small-scale features are simulated using Template 2 (Figure 4.9c). In addition to the original conditioning data, the simulated set of nodes in Step 1 is used as additional conditioning information for Step 2. In order to reduce the overwhelming influence of all these conditioning data, 50 points are randomly sampled from this expanded set of conditioning locations and the second step of the simulation is conditioned to this reduced set. Simulation is performed until 300 iteration steps are reached. The fault network obtained at the end of Step 2 is shown in Figure 4.11b. It is determined that the pattern simulation yields a simulated fault network with similar connectivity patterns as in the Valley of Fire fault map. Large and small-scale features following the main orientations are simulated. Also, curvilinear structures that are similar to the ones in the original image are also reproduced and these features are emphasized by the arrows in Figure 4.11b.

Comparing the final simulation result (Figure 4.11b) to the Valley of Fire network in Figure 4.9a, it is evident that the small-scale template added details such as parallel fault strands at certain locations along the fault. Moreover, additional intersections are simulated along the large scale faults and these features are emphasized by blue arrows on Figure 4.11b.

Further comparison of the simulation results to the original network is performed by plotting the pattern histograms. For this purpose, both the original and simulated point-set data are scanned using a 4-node template (Figure 4.12a). It is observed in Figure 4.12b that the simulated histogram is similar to the target histogram; most of the patterns are reproduced and the corresponding frequency values are in agreement.



a- Template for histogram comparison



b- Pattern Histograms

Figure 4.12- Pattern Histogram Comparison for Valley of Fire Fault Network. The template used for scanning both the original fault network as well as the simulated model is shown on the left. The resultant histograms are shown on the right.

It can be concluded based on this example that the non-gridded MPS analysis and pattern simulation algorithms are successful at characterizing and modeling networks with complex structures and multiple orientations. By applying a multi-grid approach,

features at multiple scales exhibiting different orientations are successfully reproduced. In the following section, application of the algorithm using 3D network data is presented.

4.5.2 3D Synthetic Fracture Network

Pattern simulation of the 3D synthetic fracture network data is performed using the previously inferred statistics. 4-node spatial templates and corresponding pattern histograms (Figure 4.13-4.14, Table 4.1) are implemented concurrently to simulate patterns in different orientations. Detailed analyses of these pattern histograms are given in Chapter 3.

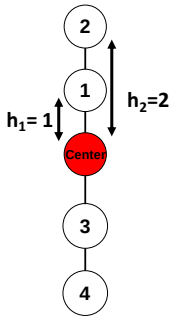
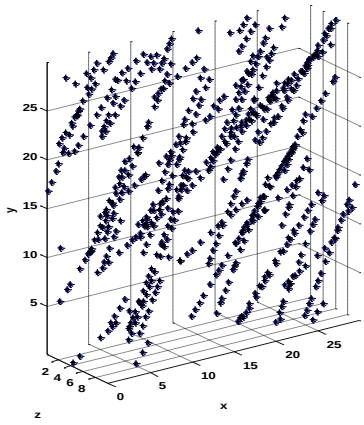


Figure 4.13- 4-Node Spatial Template used in Pattern Simulation of 3D Synthetic Fracture Network

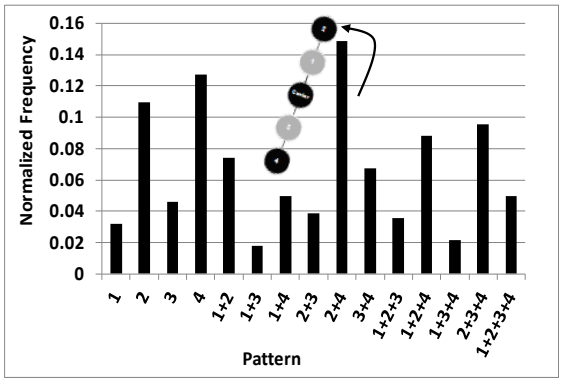
Table 4.1- Properties of the complex spatial template used for pattern simulation of the 3D Synthetic Fracture Network

Template	Lag Tolerance Factor	Azimuth Angle (°)	Dip Angle (°)
1	1/2	20	30
2	1/2	35	10
3	1/2	50	20

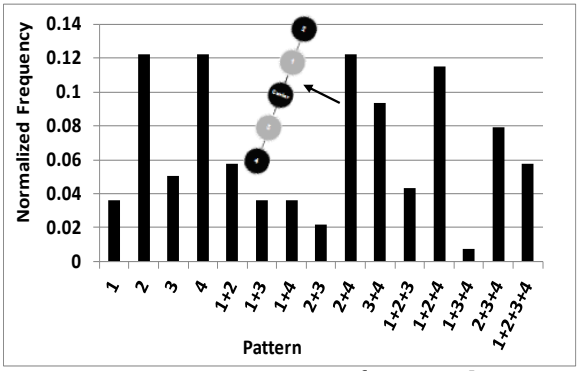
Realizations are conditioned to sparse data randomly selected from the 3D synthetic data set. To avoid data clustering, a minimum distance (5 units) between the conditioning data locations is specified.



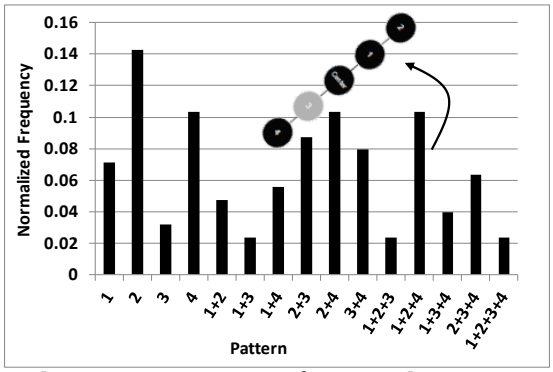
a-3D Synthetic Network



b- Pattern Histogram for Template 1



c- Pattern Histogram for Template 2



d- Pattern Histogram for Template 3

Figure 4.14- Pattern Histograms of 3D Synthetic Fracture Network

4.5.2.1 Pattern Simulation without Histogram Filtering

The first set of realizations for the 3D synthetic network data is performed using 100 conditioning data. The simulation is conducted by using the inferred statistics for Templates 1, 2 and 3 (Table 4.1), and it is terminated after 200 simulation steps. At each step of the simulation process, the possible patterns using each of the three templates are retrieved and the pattern with the highest probability is selected and applied at the simulation node, and the simulation is progressed. One of the pattern simulation results is shown in Figure 4.15.

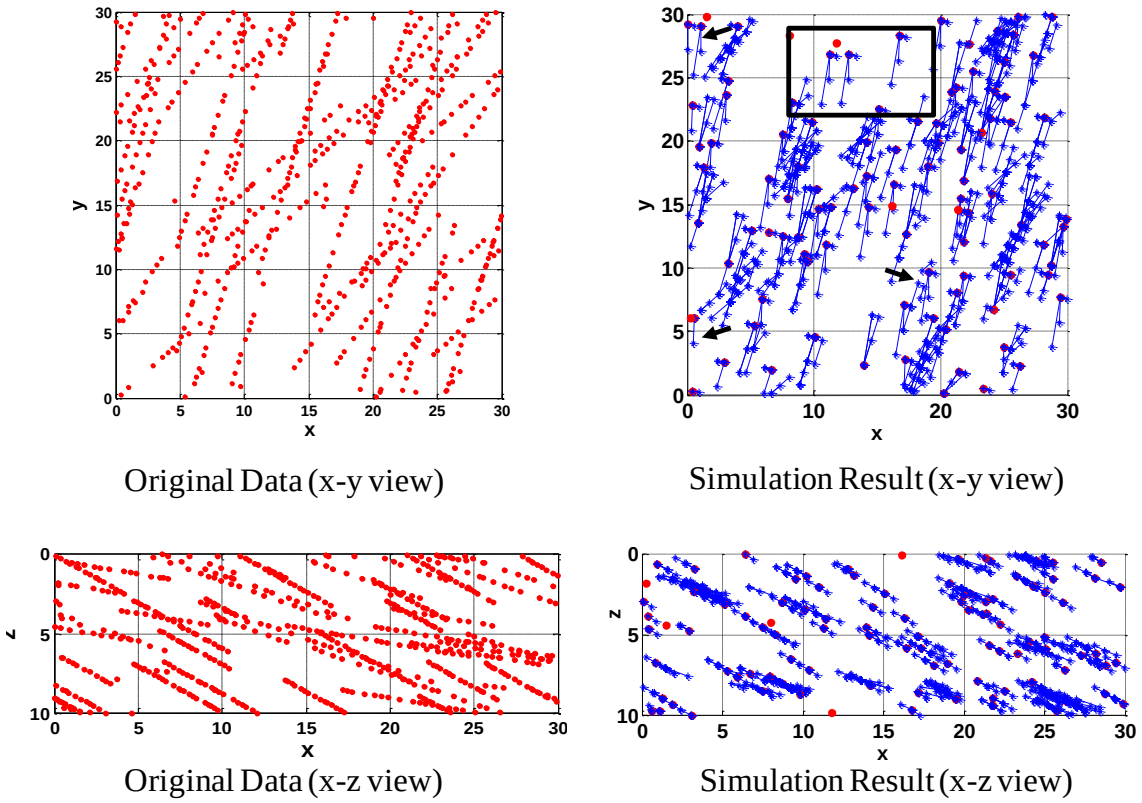


Figure 4.15- Pattern Simulation Results without Histogram Filtering for 3D Synthetic Fracture Network (Red Circles: Conditioning Data; blue *: Simulated Point). Spurious simulated connections are shown by black arrows and inside the black rectangle.

It is observed that the simulation results are consistent with the original training image and patterns of connectivity observed in the training set are satisfactorily reproduced. Although there are some spurious connections simulated (Figure 4.15), the simulated network is reasonable. Since pattern reproduction in terms of the mp histogram is crucial for accurate results, the simulated point set is scanned using a single 12- node template that is a combination of the templates implemented in pattern simulation. The multiple point histogram for the simulated realization was compared to that for the original training set (Figure 4.16). The analyses are performed by using a tolerance

window defined by 20° azimuth tolerance, 10° dip tolerance and a lag tolerance factor of 1/3.

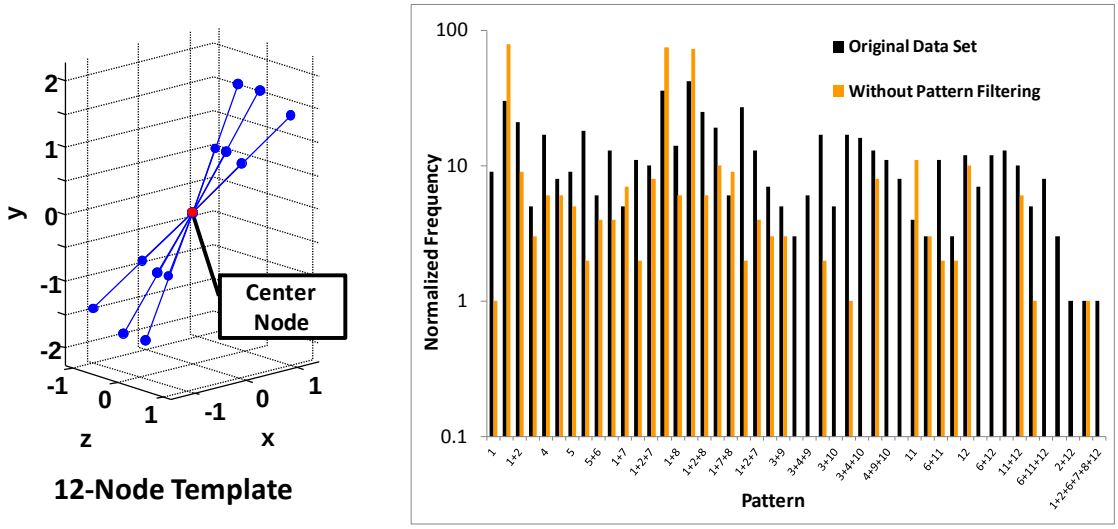


Figure 4.16- Comparison of multiple point histogram for the simulated realization and the original 3-D data set. The template used for scanning the histograms is shown on the left.

Although the pattern histogram is reproduced satisfactorily, there are significant deviations from the target mp histogram. Some of the patterns in the original data set (shown in black in Figure 4.16 right) are not reproduced while the frequency of some features is excessive in the simulated model indicating clustering of certain configurations. Moreover, there are some spurious simulated connections that are not observable in the original image (Figure 4.15). Thus, the pattern simulation algorithm is modified to overcome these problems by filtering spurious features corresponding to low pattern frequency. Simulation with the histogram filtering method and corresponding results are presented in the next section.

4.5.2.2 Pattern Simulation with Histogram Filtering

As seen from the comparison of the simulated and target images, a number of spurious features appear in the simulated model (Figure 4.15). A servo-mechanism is

implemented using which the multiple point histogram of the simulated image is checked periodically and patterns with very low frequency are eliminated. Histogram filtering is applied after every 100 simulation step. This set of realizations is obtained using 50 conditioning data locations, 400 simulation steps and a low-frequency threshold value of 5 (Figure 4.17). The number of conditioning points is reduced to demonstrate the capability to reproduce patterns with less conditioning data. Moreover, the simulation is terminated at 400 steps (unlike the previous results obtained by 200 simulation steps) for better application of pattern filtering. The same templates and pattern histograms used previously are also utilized for this set of realizations.

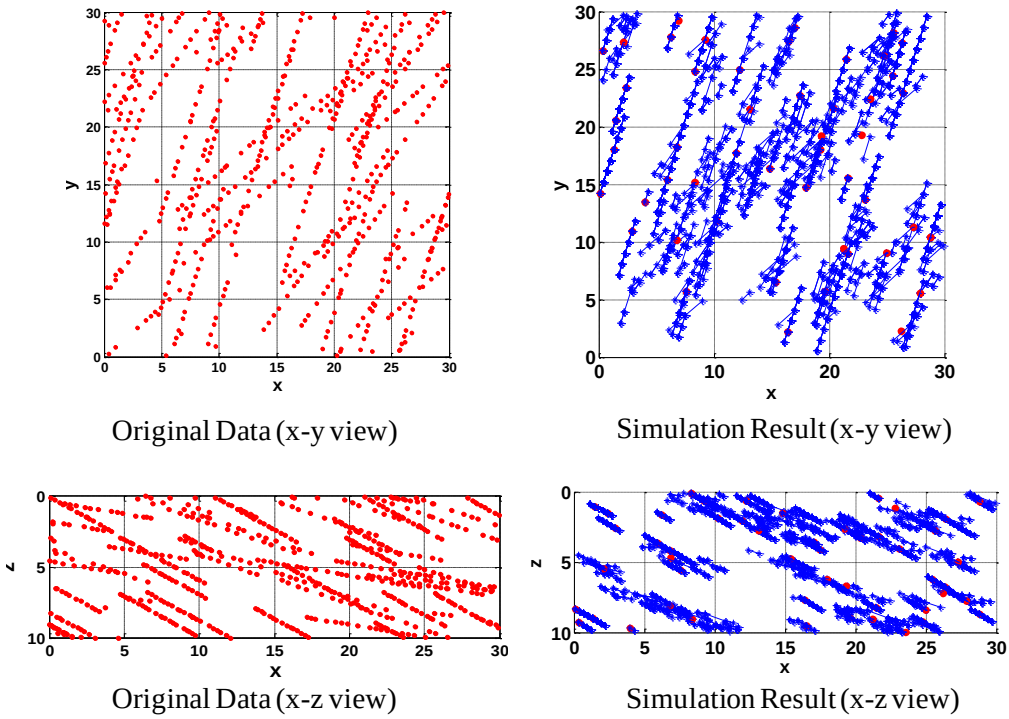


Figure 4.17- Pattern Simulation Results with Histogram Filtering for the 3D Synthetic Fracture Network (Red Circles: Conditioning Data; blue *: Simulated Point). The patterns observed in the original training network are shown in the left while the simulation result is shown on the right. X-Z views are along y=0 line.

By applying histogram filtering in the simulation algorithm, the number of spurious simulated connections is reduced. In Figure 4.18, improved simulation results are compared to the previously simulated image. It is observed that the histogram filtering eliminates spurious connections and yields more accurate results.

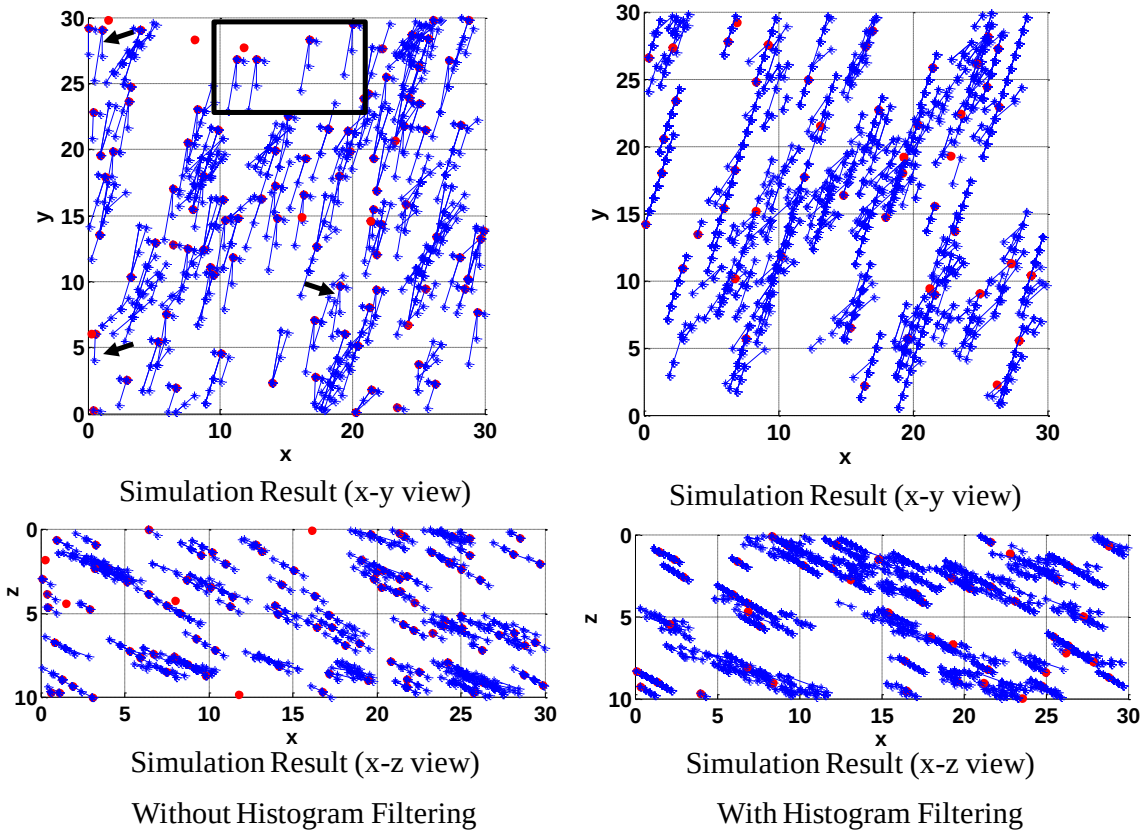


Figure 4.18- Comparison of Simulation Results with Effect of Histogram Filtering. Spurious features are shown by black arrows and black rectangle.

Similar to the analysis conducted for the previous set of realizations, the simulated network is scanned using the 12-node template (Figure 4.16) using the some tolerance window, and the calculated MP-histograms are compared (Figure 4.19).

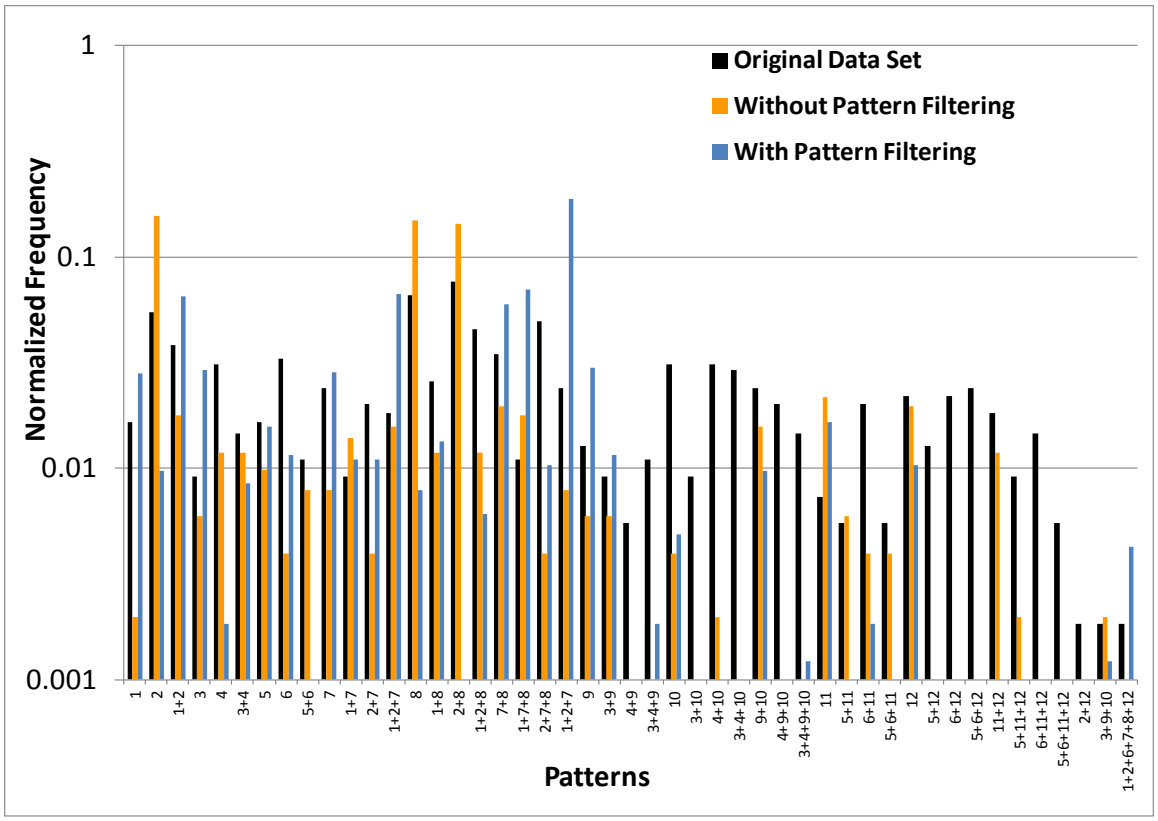


Figure 4.19- Pattern Histogram Comparison (Simulation with Histogram Filtering). The histogram corresponding to the original data (black), simulation without histogram filtering (orange) and simulation after histogram filtering (blue) are compared. Semi-log plot is used for better visual comparison.

Results confirm that better representation of spatial patterns is possible using the pattern-filtering algorithm. Histogram filtering yields more accurate results in terms of pattern reproduction and pattern frequency distribution. Compared to the previous set of results, patterns that were not previously reproduced start to appear in the histogram. It is concluded that this servo mechanism of eliminating spurious connections results in accurate representation of network connectivity. Although the results are satisfactory, another set of realizations is obtained for the 3D synthetic fracture network using a sequential approach to feature modeling. The following section presents the results for this sequential approach.

4.5.2.3 Sequential Feature Simulation

Results shown in the previous sections are obtained by concurrently implementing 3 different templates during the simulation. Although the previous techniques are successful in pattern and connectivity reproduction, certain configurations are dominant in the simulation results. Since patterns are selected from the histogram based on the probability, configurations with higher frequency are preferentially applied during the simulation. This may result in oversampling of some features. In order to prevent this sampling bias, the selected templates are sequentially implemented for simulating one set of features at a time. The templates whose properties are given in Table 4.1 are implemented in the simulation.

In the first step, 4-node template with 20° azimuth and 30° dip angles is implemented. This first step is conditioned to 50 data points sampled from the original training network. The simulation is terminated after 200 steps. The simulated points are utilized as input for the second step. The second feature simulation is performed using a 4-node template with 35° azimuth and 10° dip angles. 50 conditioning data points are randomly sampled from the first step simulation. The third template is specified in 50° azimuth and 20° dip orientation, and the corresponding pattern histogram is applied. 50 conditioning data points are sampled from the results of the second feature simulation.

At the end of sequential feature simulation, a realization of the 3D synthetic network is obtained (Figure 4.20). Visual comparison to the original data suggests that the sequential feature simulation results in accurate and representative patterns. Clustering of patterns due to preferential selection of patterns corresponding to the highest probability is reduced. Moreover, the simulated feature lengths are consistent with the original data set.

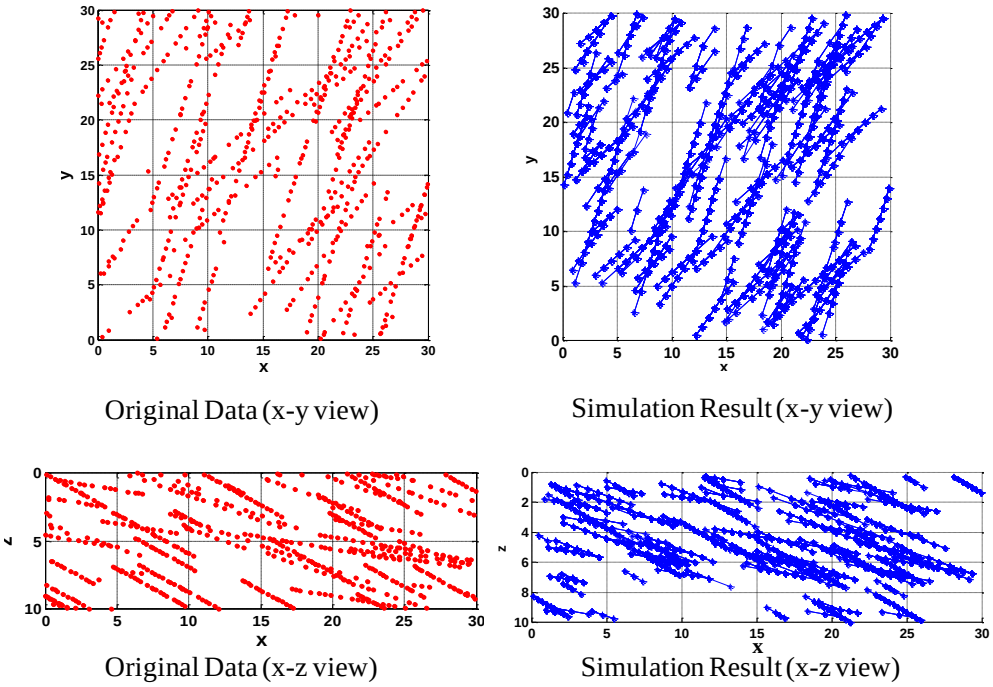


Figure 4.20- Sequential Feature Simulation of the 3D Synthetic Fracture Network. The three templates shown in Figure 4.13 and Table 4.1 are used for scanning and simulating the patterns shown on the right.

Comparison of the simulation results to the original network is performed by scanning the simulated network using the 12-node template using the same tolerance window shown in Figure 4.16. The resultant MP-histograms are presented in Figure 4.21.

It is observed that almost the entire original histogram is successfully reproduced. By implementing sequential feature simulation, results are improved in terms of pattern reproduction and pattern connectivity. Visual comparison suggests that the clustering of certain configurations at certain locations and spurious connections are also eliminated (Figures 4.22). Moreover, larger sized features in the main orientations are successfully reproduced.

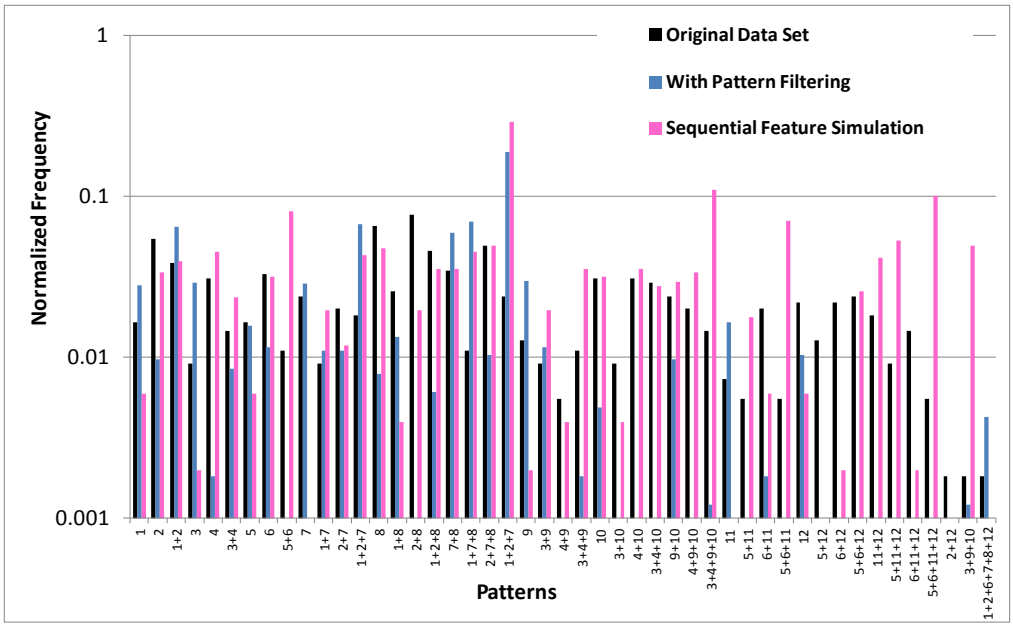


Figure 4.21- Comparison of Pattern Histogram corresponding to Sequential Feature Simulation (pink), original network (black) and the previous simulation by filtering the pattern statistics (blue).

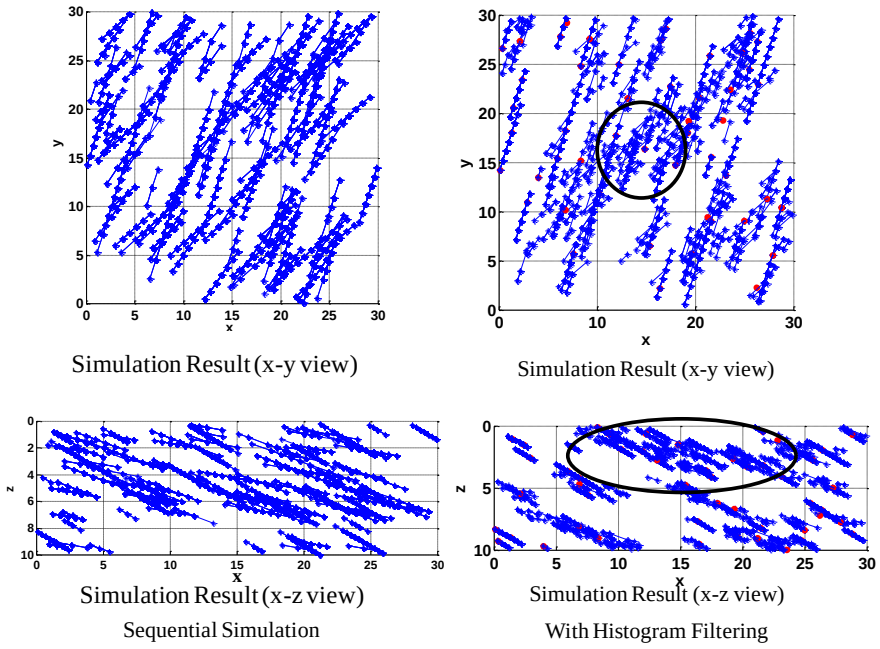


Figure 4.22- Comparison of Simulation Results with Sequential Pattern Simulation (left) and with Histogram Filtering (right). Clustering of features that are shown in the marked regions is eliminated by the sequential simulation.

In conclusion, non-gridded MPS analysis and pattern simulation algorithms are successful at characterizing and modeling connected features for which information is available only in the form of point sets. Applicability of the proposed algorithm on 2D and 3D network systems are presented by utilizing different simulation options. In order to model the spatial distribution of cave facies in subsurface, it is essential to simulate cave openings represented by as cave zone thickness. Therefore, an algorithm is developed to simulate cave opening/cave zone thickness. This algorithm is embedded into the pattern simulation algorithm to concurrently obtain network and corresponding cave zone thickness along the simulated passages. In Chapter 5, cave opening simulation algorithm is presented.

Chapter 5: Modeling of Cave Opening

Cave facies have an important effect on fluid flow. As described earlier in this dissertation, cave facies result from the collapse of caves and to get an accurate model for the spatial extent of the cave facies, it is essential to model the width of the cave opening. Therefore, an algorithm is developed to simulate cave opening/cave zone thickness while simulating the cave/fracture network. Since the network growth starts at the conditioning data locations, cave width values at these locations and a variogram model that describes the variation in cave width are required. For a paleokarst reservoir, thickness of cave facies zone is implemented instead of the cave width because these paleocaves are rarely open. The thickness of the cave facies zone is interpreted based on well logs. Strategies to identify cave facies zone are discussed in Chapter 7.

5.1 Simulation Algorithm

The simulation algorithm discussed in subsequent sections can handle width variations described using normal and log-normal distributions that have to be user specified. Since the conditioning data might follow a log-normal distribution, these values are transformed first into a normal distribution before network simulation starts. Thus, at the end of the pattern simulation, the simulated cave zone thicknesses should be back transformed to the log-normal space. In order that the conditioning data are preserved, the estimation variance values are also recorded. This back-transformation can be accomplished using the *trans program in* SGeMS (Remy et al., 2008) that uses the estimation variances as a measure of proximity to data.

Once the user provides the conditioning cave zone thickness values and the variogram model, the thickness values are calculated at the simulated nodes and along the simulated passages obtained using the MPS algorithm. The cave zone thicknesses are simulated using Sequential Gaussian Simulation (SGSIM) algorithm with ordinary

kriging modified such that the thicknesses are simulated concurrently with the MPS simulation. The cave zone thickness simulation does not affect the pattern simulation algorithm; it is only performed when a new "simulatable node" is calculated and marked as a cave location.

At the first simulation step, the entire set of conditioning data locations is utilized while calculating the kriging system. As the conditioning data set (original conditioning + previously simulated nodes) grows during the simulation, data locations within a search range are included in kriging. A user-defined maximum number of included data points is also assigned in order to avoid extremely large kriging matrices for efficient computation purposes.

5.2 IMPLEMENTATION DETAILS

Once the cave zone thicknesses at two nodes along a cave path have been simulated, linear interpolation is performed between the beginning and the end nodes of the passage. The passage is divided into several increments and cave zone thickness is calculated by linear interpolation at the inter-passage node (Figure 5.1). Thus, a cave opening map is constructed along the simulated network.

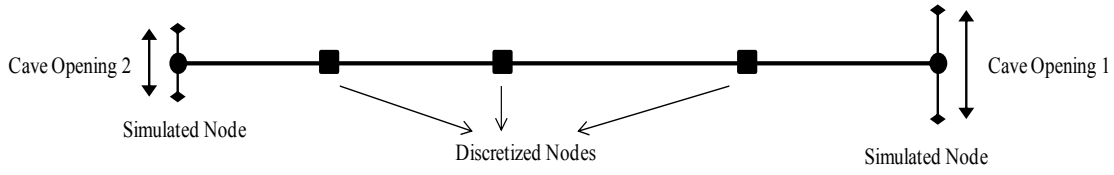


Figure 5.1- Discretized Simulated Passage. Passage beginning and end nodes are shown as simulated nodes. Cave opening at the discretized nodes are calculated by linear interpolation between Cave Opening 1 and Cave Opening 2.

The algorithm for modeling cave zone thickness is an effective tool to simulate the spatial extent of the cave facies while simulating the cave network. It enables the user to integrate information from various sources for the simulation of the subsurface

paleokarst network. Using this tool, one can obtain a paleokarst network with associated paleokarst zone thicknesses to integrate into the flow simulation models. Moreover, this algorithm does not require gridding of the simulation domain as cave zone thickness is simulated using only the point set data.

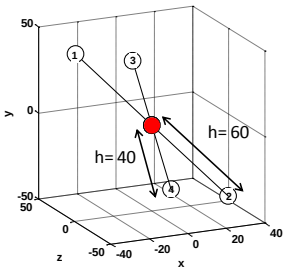
Cave zone thickness should be provided as an input to the simulation. In modern caves, this information might in the form of measured cave width or 3D model of the cave network with central line information. In case of a paleokarst reservoir, cave facies zone is determined at the well locations by using well log data. First, wells with recognized cave facies are identified and then, gross cave zone thicknesses along the wellbore are determined. Once the cave zone thicknesses are obtained, a variogram is calculated and the corresponding model is implemented in the pattern simulation algorithm. Details of an approach for cave facies recognition and its application to Yates Field are presented in Chapter 7.

5.3 AN IMPLEMENTATION EXAMPLE

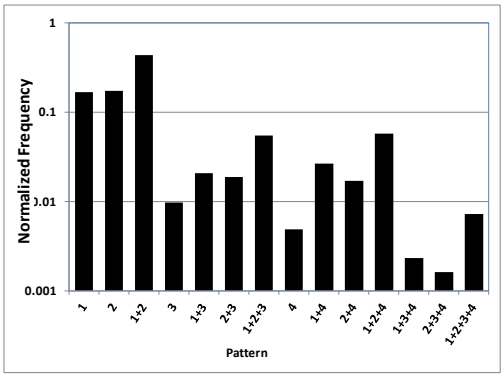
For demonstration of the cave opening simulation algorithm, a modern cave network is simulated using a synthetic cave width data set. 3D cave central line plot of the Wind Cave in South Dakota is available and it is considered for this demonstration example. Although detailed MPS analysis and pattern simulation of the Wind Cave are given in the following chapters, an example of a simulated cave network and corresponding cave opening simulation for a sub region of Wind Cave are presented in this chapter.

Non-gridded MPS analysis and cave network simulation of Wind Cave are performed for Region 2. The basis for defining this sub-region is discussed in detail in the subsequent chapters. The point set data for Region 2 is used as a training image. Cave network simulation is performed in two steps; first, large scale structures are simulated

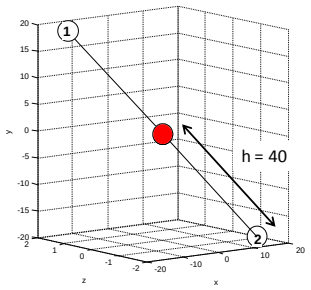
using a 4-node spatial template and then small scale features are obtained by a smaller sized template. In Figure 5.2, these templates and MP-histograms constructed by scanning the Region 2 point set data are shown. For both templates, the tolerance window is defined by an azimuth tolerance of 20°, dip tolerance of 10° and a lag tolerance factor of 1/3. Details of template selection and further comparison of MP-histograms are given in Chapter 6.



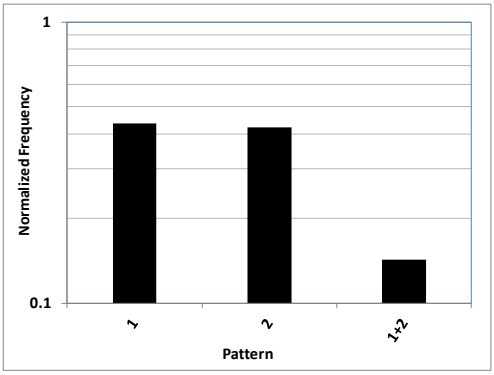
a- Template 1
Azm= -40°; Dip= 5°, 89°



b- Pattern Histogram of Template 1



c- Template 2
Azm= -40°; Dip= 5°



d- Pattern Histogram of Template 2

Figure 5.2- Templates and Pattern Histograms used in Network Simulation of Region 2, Wind Cave. Details of template selection and further comparisons are given in Chapter 6.

After construction of MP-histograms, the inferred statistics are implemented in pattern simulation algorithm. 100 conditioning data points are sampled from the Region 2

point set for the cave network simulation. At this step, a synthetic cave opening data set consisting of 100 points that are randomly sampled thickness in the 10 ft -50 ft range is implemented. The probability distribution exhibited by this synthetic cave opening data set is given in Figure 5.3.

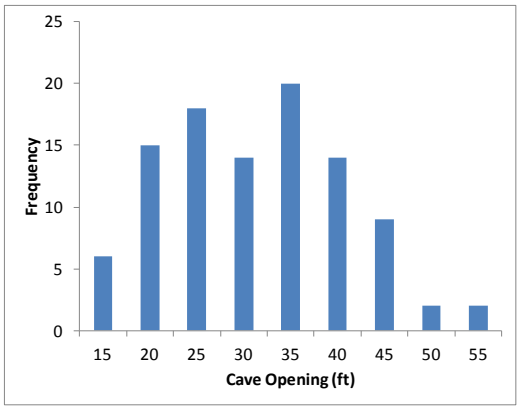
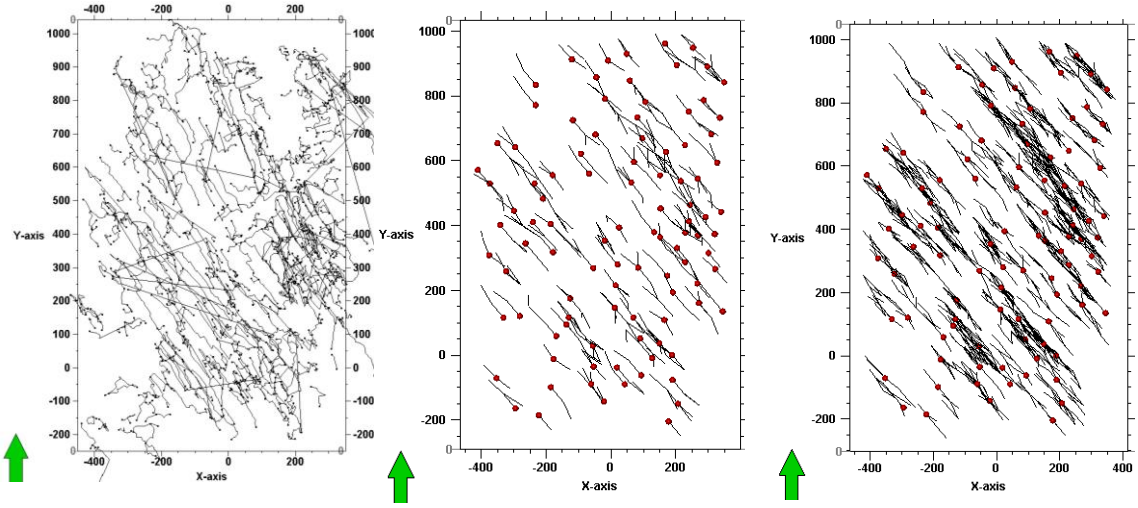


Figure 5.3- Histogram of Synthetic Cave Opening Data Set

Pattern simulation is performed using the templates and MP-histograms given in Figure 5.2. The simulated cave network is presented in Figure 5.4. It is observed that the final simulated model (Figure 5.4c) is successful in representing the original network.



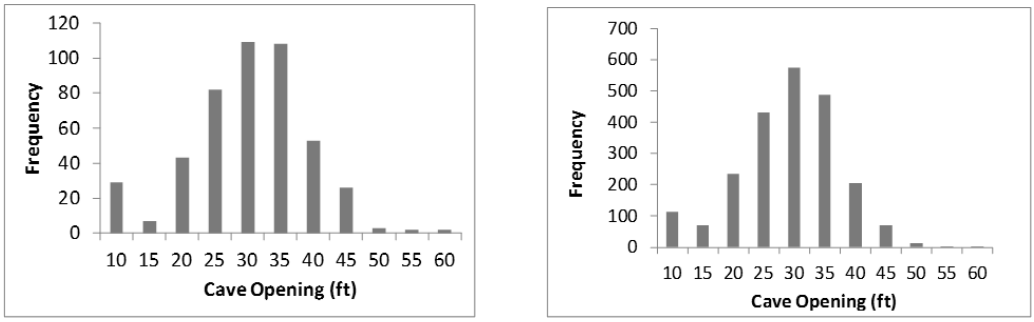
a- Region 2 Cave Network

b- Pattern Simulation of Large Scale Features

c- Pattern Simulation of Mid Scale Features

Figure 5.4- Pattern Simulation Results. The original Wind Cave data and the simulated model are compared (Red points: Conditioning Data; Black Lines: Simulated Connections. Arrow shows the North direction.)

The cave opening simulation is concurrently performed with the network simulation. For this demonstration exercise, cave width of Region 2 is simulated using the conditioning data set shown in Figure 5.3. An exponential variogram with a range of 100 ft and a sill contribution of 1 is implemented for the application. The simulated cave opening data at Step 1 and Step 2 are presented in Figure 5.5. It is observed that the simulated distribution is similar to the original distribution (Figure 5.3). Also, statistics of the simulated distributions are compared to the original values (Table 5.1) and it is concluded that simulation algorithm is successful at honoring the original cave opening distribution.



a- Simulated Cave Opening Distribution (Step 1) b- Simulated Cave Opening Distribution (Step 2)

Figure 5.5- Simulated Cave Opening Distributions

Table 5.1- Statistics of Cave Opening Distributions.

	Mean (ft)	Median (ft)	Std. Deviation (ft)
Original Set (Figure 5.3)	29.0	28.3	9.7
Simulation Step 1 (Figure 5.5a)	27.5	28.5	10.1
Simulation Step 2 (Figure 5.5b)	26.5	27.7	9.1

Simulated cave opening along a passage at the end of Step 2 is illustrated in Figure 5.6. At the simulated nodes, cave opening is calculated by SGSIM whereas at the discretized nodes, it is obtained by linear interpolation between the simulated nodes of the passage.

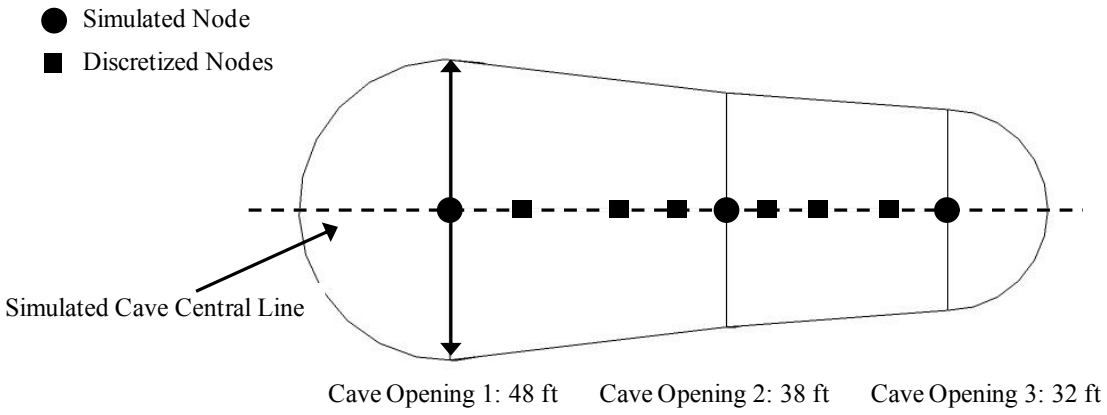


Figure 5.6- Simulated cave opening along a passage at the end of Step 2.

In this chapter, a cave opening/cave zone thickness simulation algorithm is presented and it is demonstrated for simulating the cave network of the Wind Cave. A synthetic data set was used to condition the simulation. In Chapter 6, detailed MPS analysis and pattern simulation results for Wind Cave are presented.

Chapter 6: Network Modeling of Wind Cave

In this chapter, application of the non-gridded MPS analysis and pattern simulation algorithms using the Wind Cave data set is presented. Wind Cave located in South Dakota was discovered in 1881 and the initial exploration surveys of the cave were conducted in 1902 (Horrocks and Szukalski, 2002). It is one of the largest cavern systems in the world (Miller, 1989) and Wind Cave currently has a total survey area exceeding 166 km of total linear passage distance (Horrocks and Szukalski, 2002). This modern cave system is formed in the Madison limestone with major trend in N52ºE slightly plunging to the northeast and with a 4°-5.5° slope to the southeast (Figure 6.1) (Horrocks and Szukalski, 2002). Earlier studies proposed that the cave system consists of mazes that are developed by upward movement of groundwater (Ford, 1989). More recently, Horrocks and Szukalski (2002) concluded that cave passages and sinkholes were formed in a mixing zone environment of sea and meteoric waters.

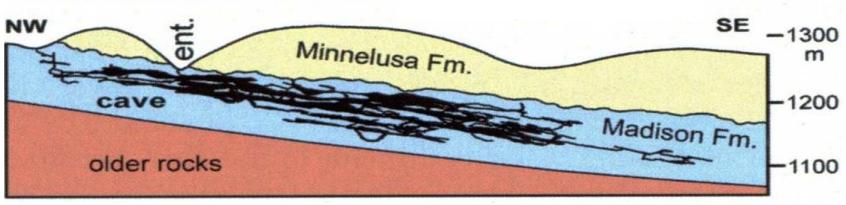


Figure 6.1- Cross-Section of Wind Cave (Palmer and Palmer, 2008)

The 3D cave central data is available for Wind Cave and a line plot of the data is presented in Figure 6.2. The line plot consists of data point coordinates and includes point connection information reporting the connectivity of each node to the neighboring ones.

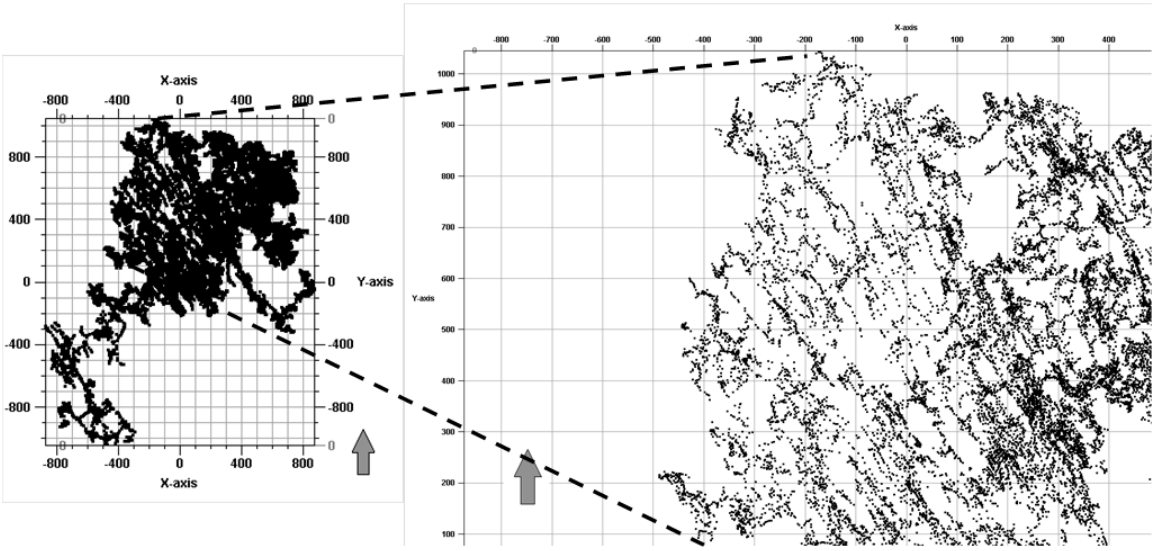
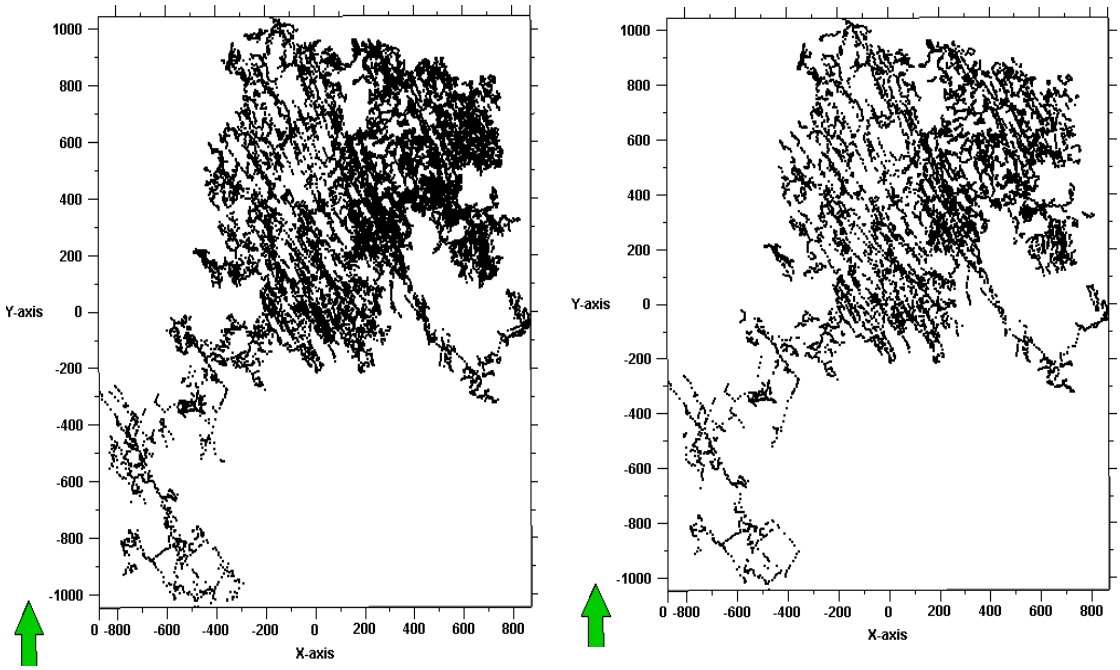


Figure 6.2- Line Plot of Wind Cave Central Line (Arrow shows the North direction.)

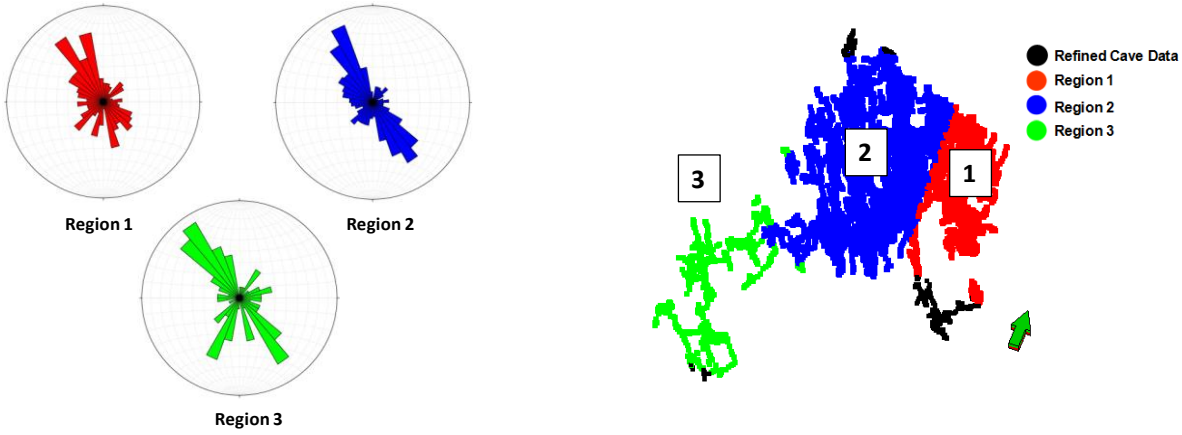
Since the cave has a large surveyed areal extent, it is practical to perform MPS analysis on a subset of the complete data set and later using the inferred statistics to simulate the entire cave system. After initial examination of the Wind Cave galleries, some short and circular cave segments are eliminated (Figure 6.3). Then, the remaining cave passages are divided into 3 main regions based on the visually observed main trends and orientations of the cave passages are examined for each region. Because stationary domains are a must for statistical inference, passage orientations in the divided regions are calculated for comparison. First, azimuth orientations of the passages are determined and a rose diagram is constructed for each region using 10° bins (Allmendinger, 2011). Rose diagrams are circular histograms of directional data and are commonly used for analyzing fracture orientations. Thus, azimuth angle distribution of cave passages in each region is given by the corresponding rose diagram; if the rose diagram shows a single orientation, that particular region is considered as stationary. These analyses are performed until Wind Cave is divided into 3 stationary regions and corresponding rose

diagrams are obtained (Figure 6.4). After the region division, MPS analysis of the Wind Cave is performed on these stationary point-set training images.



Original Cave Data Reduced Point Set

Figure 6.3- Original Cave Data and the reduced point set.

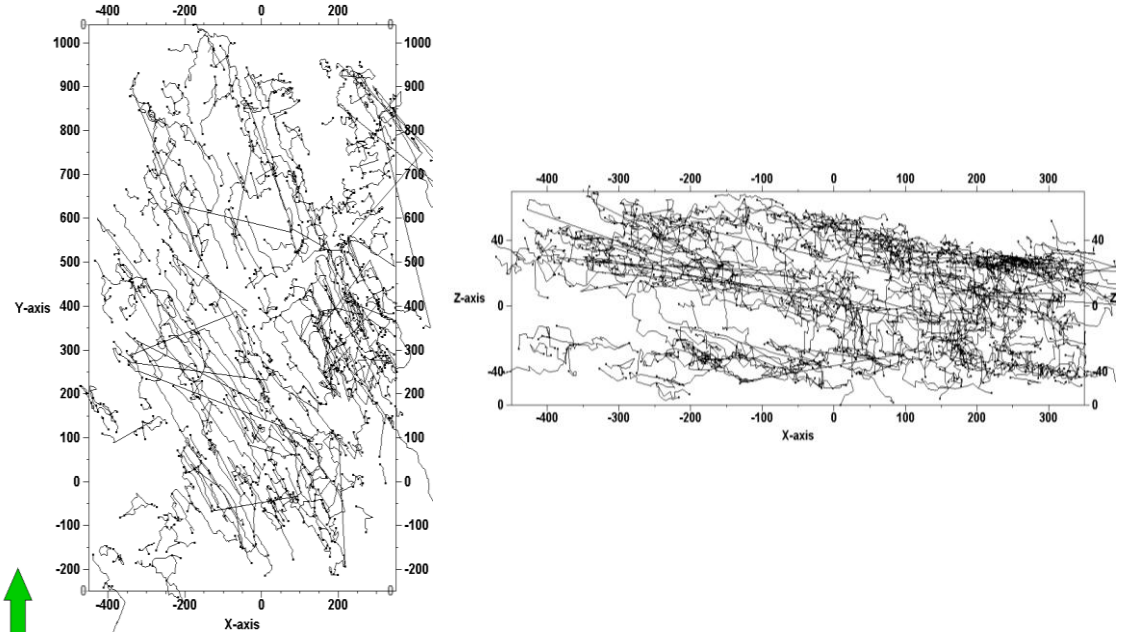


a- Rose Diagrams b- Cave Sub Regions

Figure 6.4- Cave Passage Rose Diagrams and Sub Regions

6.1 NON-GRIDDED MPS ANALYSIS OF WIND CAVE

It is observed that the majority of the cave passages are in Region 2 and they follow a preferential orientation (Figure 6.4a). Also, Region 2 has relatively uniform cave branching with passages along two main orientations (NW-SE and NE-SW) (Figure 6.5) and it is an ideal candidate for MPS analysis. Thus, Region 2 is selected as the non-gridded training image to infer the pattern statistics of Wind Cave (Figure 6.5). The point set training image has 10,729 data locations and due to computation limitations, a refined set of 6,000 points is randomly selected in MPS analysis. Thus, each template calculates MP-statistics on a different refined set of 6,000 points.



a- Top View b- Cross-Section

Figure 6.5- Cave center line plot of region 2 in top view and cross-section view. This region is used as the non-gridded training image.

Preliminary analysis of Region 2 was performed using generic templates. First, a set of 19-node generic templates is constructed by assigning nodes at a succession of azimuth angles spanning a 90° range. For each generic template, a different dip angle is

assigned varying from 0° to 90°. Then, the point-set training image of Region 2 is scanned using these generic templates and the number of connections captured by each template is computed. The properties of generic templates are given in Table 6.1; positive azimuth direction indicates that the template nodes are oriented in NE direction whereas negative azimuth direction is assigned for nodes in NW direction. Lag distances of generic templates are assigned based on average passage length that is obtained by examining the 3D network model of Region 2 (Figure 6.5). Initial MPS analysis shows that the maximum number of connections is captured in the 5° to 10° dip direction. This prediction is accurate since the Madison limestone hosting Wind Cave is dipping at a 5° angle (Horrocks and Szukalski, 2002).

Table 6.1- Properties of the most significant generic templates for Region 2

Template	Lag Distance (units)	Azimuth Direction	Dip Angle (°)	# of Connections Captured
Generic 1	25	0° to 90°	5	1559
Generic 2	25	0° to 90°	10	1205
Generic 3	25	0° to 90°	60	420
Generic 4	25	0° to 90°	89	213
Generic 5	25	0° to -90°	5	2020
Generic 6	25	0° to -90°	60	566

After the initial generic template analysis, multi-node templates along the predicted main orientations are constructed to capture the spatial distribution of continuous passage as accurately as possible. Eight different 16-node templates are defined with nodes at various lag distances and 5° dip angle (Figure 6.6). These long templates enable one to understand the average length and maximum extent of continuous passages in the network. By constructing them in several azimuth directions and adjusting the azimuth angle tolerance, these multi-node linear templates are able to capture connections in various orientations at different length scales with the use of

multiple nodes assigned at succession of 10 units. Properties of the templates and the number of connections captured are presented in Table 6.2.

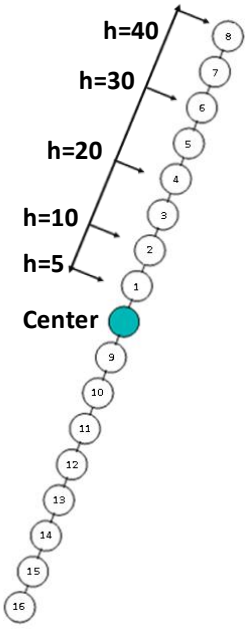


Figure 6.6- 16-Node Template for Region 2

Table 6.2- Properties of 16-Node Templates for Region 2

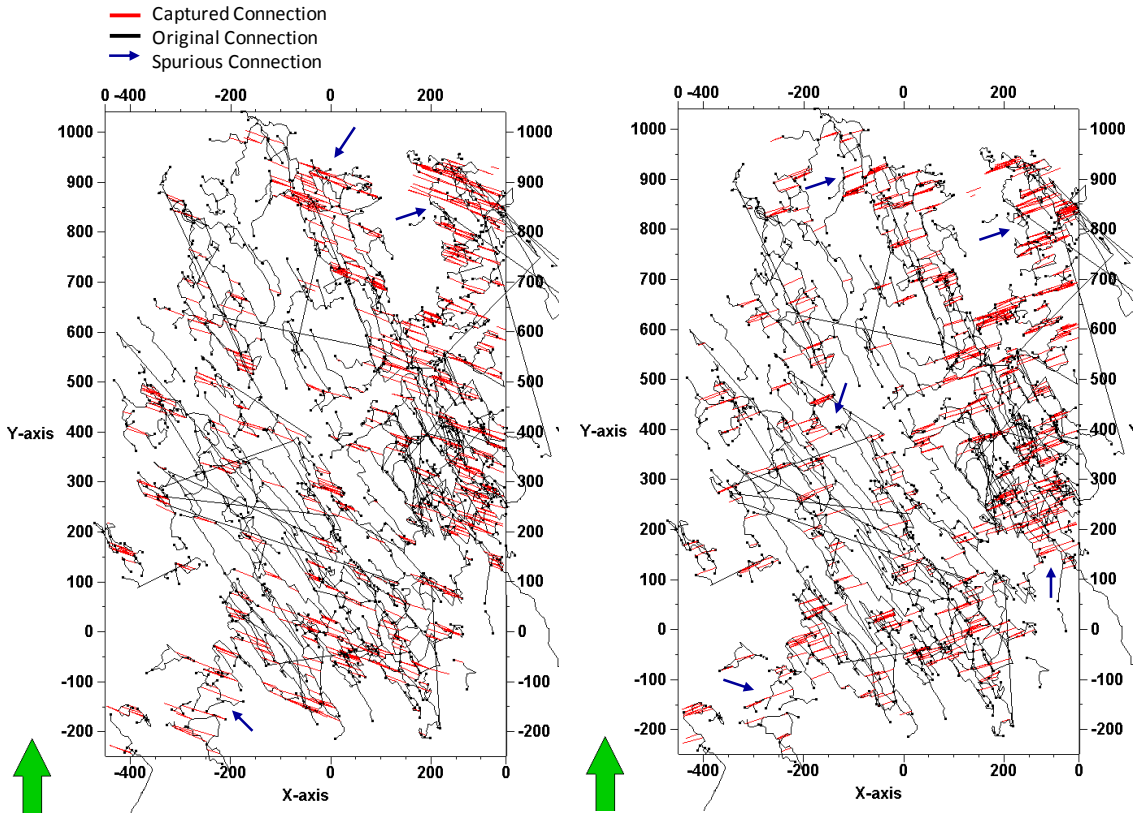
Template	Lag Tolerance Factor	Azimuth Angle (°)	Azimuth Angle Tolerance (°)	Dip Angle (°)	Dip Angle Tolerance (°)	# of Connections Captured
L-1	1/3	70	10	5	10	2034
L-2	1/3	40	10	5	10	929
L-3	1/3	20	10	5	10	577
L-4	1/3	50	10	5	10	2034
L-5	1/3	-70	10	5	10	2573
L-6	1/3	-40	10	5	10	3230
L-7	1/3	-20	10	5	10	1587
L-8	1/3	-50	10	5	10	2864

It is observed that 16-node templates with azimuth angle of 50°, 70°, -40°, -50° and -70° capture significant number of connections. Corresponding pattern histograms show that large-scale features are captured by nodes at a lag spacing h=40 units whereas

smaller scale passages are better represented using nodes that are separated by a lag distance of 20 to 30 units.

Since large-scale templates might capture spurious connections, the validity of the template orientation should be verified by visual comparison with the training image. Thus, connections captured by templates L-1 and L-5 are examined and it is determined that these templates mostly report spurious connections (Figure 6.7). Although initial generic template analysis reports significant amount of connections in 70° and -70° azimuth directions, spurious connections are captured by using these templates at larger scale. Therefore, spatial templates should be selected carefully and these spurious connections could be reduced by further tightening the tolerances. As a result, templates L-1 and L-5 are excluded for further MPS analysis.

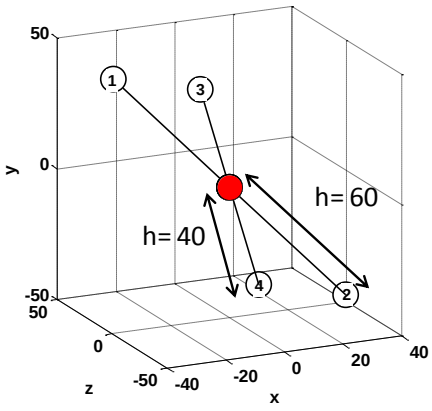
A 16-node template enables identification of large and small-scale network structures. This template is used mainly to gain an understanding of the cave passage lengths that can subsequently used for specifying the lag distances for the subsequent complex spatial templates.



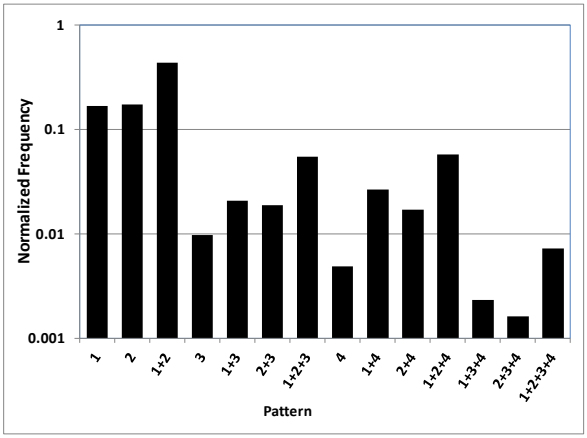
a- Template L-1 b- Template L-5

Figure 6.7- Connections Captured by Templates L-1 and L-5. Spurious connections are shown by blue arrows. Green arrow shows the North direction.

Three different spatial templates are assigned for capturing features at large scale, mid scale and small scale. In these analyses, tolerance window is defined by an azimuth tolerance of 20°, dip tolerance of 10° and a lag tolerance factor of 1/3. The first template is a 4-node template in -40° azimuth direction (Figure 6.8a). Two of the template nodes are oriented along 5° dip angle (nodes 1 and 2) and, nodes 3 and 4 are almost in vertical. The vertical nodes are to capture vertical connections between the large scale cave passages. The point set training image of Region 2 is scanned using Template 1 and MP-histogram is calculated (Figure 6.8b).



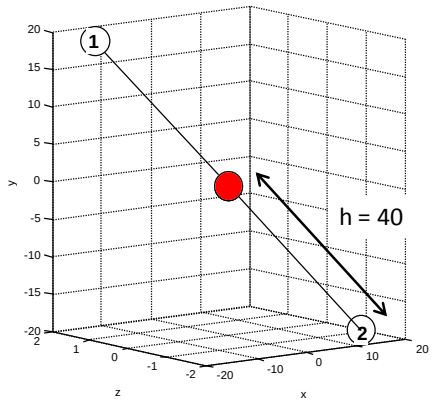
a- Template 1
Azm= -40°; Dip= 5°, 89°



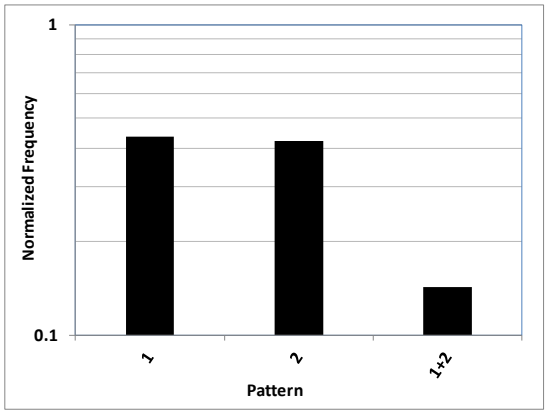
b- Pattern Histogram

Figure 6.8- MPS Analysis for Wind Cave using the Large Scale Template.

The second template is constructed to obtain connections at mid scale. The mid scale template is a 3-node template oriented along the major trend of -40° azimuth direction with a dip angle of 5° and lag distance of 40 units (Figure 6.9a). The corresponding pattern histogram is given in Figure 6.9b.



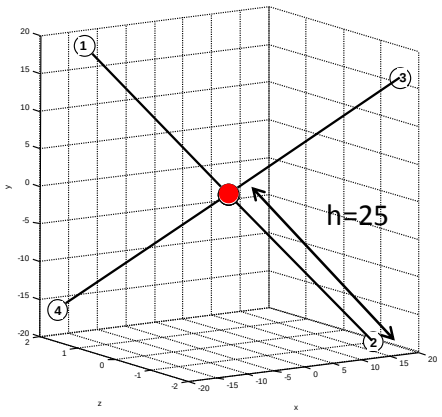
a- Template 2
Azm= -40°; Dip= 5°



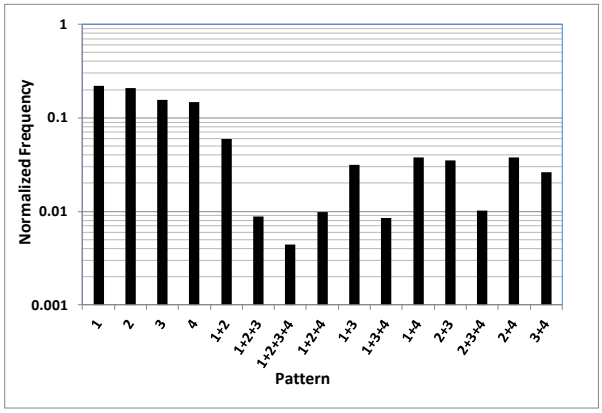
b- Pattern Histogram

Figure 6.9- MPS Analysis for Wind Cave (Mid Scale Template)

For small scale features, Template 3 is constructed as a 4-node template with lag distance of 25 units and template nodes are oriented in -40° and 50° azimuth directions with a dip angle of 5° (Figure 6.10a). The computed MP-histogram is presented in Figure 6.10b and it shows that the single node configurations (i.e. patterns 1, 2, 3 and 4) and multiple nodes forming "T" or "L" shaped configurations (i.e. patterns 1+2+3, 1+2+4, 1+4, 2+4 etc.) are most commonly observed. This observation is consistent with the training image (Figure 6.5) since there are lateral connections between the main passages and these connections follow 50° azimuth angle orientation.



a- Template 3
Azm= -40°,50°; Dip= 5°



b- Pattern Histogram

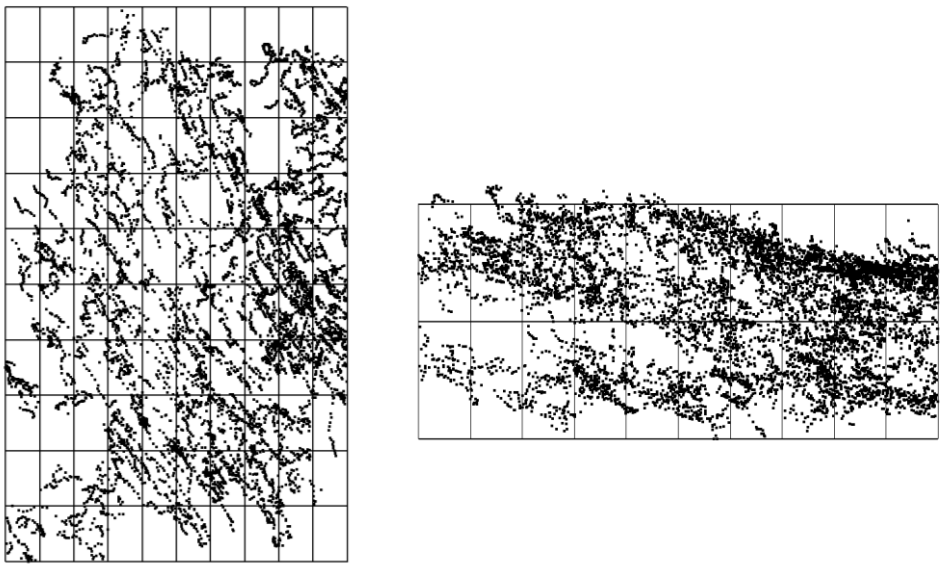
Figure 6.10- MPS Analysis for Wind Cave (Small Scale Template)

Non-gridded MPS analysis of Region 2 is performed first using 19-node generic templates in order to perform a preliminary screening of the training image and identify main orientations (Table 6.1). Then, 16-node large sized templates are constructed following the major trends to determine an average size of the continuous cave passages (Table 6.2). These steps yield valuable information for the design of 3 different spatial templates that are constructed to capture pattern connectivity of large scale, mid scale and

small scale features (Figures 6.8-6.10). Thus, pattern histograms representing the multi-scale connectivity features of the point-set training image are calculated and these MP-histograms are utilized in network simulation of Region 2. In the following section, network modeling of Region 2 is presented.

6.2 NETWORK MODELING OF REGION 2

Pattern simulation is performed using a multi-grid approach utilizing the spatial templates whose MP-statistics are previously calculated. For accurate modeling of cave network, quadrant analysis of Region 2 is performed to understand the spatial distribution of cave passage density by broadly screening the point set training image. Quadrant analysis is conducted by calculating the number of cave nodes present in a particular area (quadrant) and corresponding quadrant histogram is constructed. This histogram reports the density of cave nodes in each quadrant. Therefore, Region 2 is divided into 10*10*2 quadrants and number of nodes in each of them is determined. Figure 6.11 shows the boundary of the quadrants and in regions where the cave nodes are abundant, quadrant frequency is higher. The quadrant frequency data is implemented in the pattern simulation to serve as a control mechanism for selecting "simulation node" while simulation continues. During the simulation, if the number of simulated nodes in a particular quadrant exceeds the original quadrant frequency, "simulatable nodes" from that area will be eliminated. Therefore, cave network will stop growing from the quadrants with lower cave node density.



Top View Cross Section

Figure 6.11- Quadrant Boundaries of Region 2. Region 2 is divided into 10*10*2 small quadrants and quadrant frequency is higher in regions with more points.

Pattern simulation is conditioned to sparse data that are randomly sampled from Region 2. Using a multi-grid approach, pattern simulation is performed at 3 different steps by utilizing spatial templates at large scale, mid scale and small scale. The simulated point set at the end of each scale of simulation is implemented as input for the consequent steps.

In the first step, Template 1 and corresponding pattern histogram (Figure 6.8) are implemented for modeling large-scale passages. Simulation is performed by randomly sampling 100 conditioning data with a minimum separation distance of 75 units between them. A tolerance window is assigned around template nodes by specifying the azimuth angle tolerance of 20° at both side of the template node, dip angle tolerance of 15° and lag tolerance factor of 1/3. The simulation is continued until the maximum number of

simulation steps of 2000 is reached. The result of Step 1 is illustrated in Figures 6.12-6.13.

By comparing the simulated network to the original one, it is observed that pattern simulation successfully reproduced large scale passages following the main trend and vertical connections between the passages at different depths. Moreover, the curvilinear nature of the cave network is accurately simulated and this indicates that the utilized tolerance window is able to represent variations in network orientation. At the end of pattern simulation Step 1, a set of simulated 425 points and their connections to each other are obtained. These simulated nodes are set as conditioning data for modeling of mid scale features in Step 2.

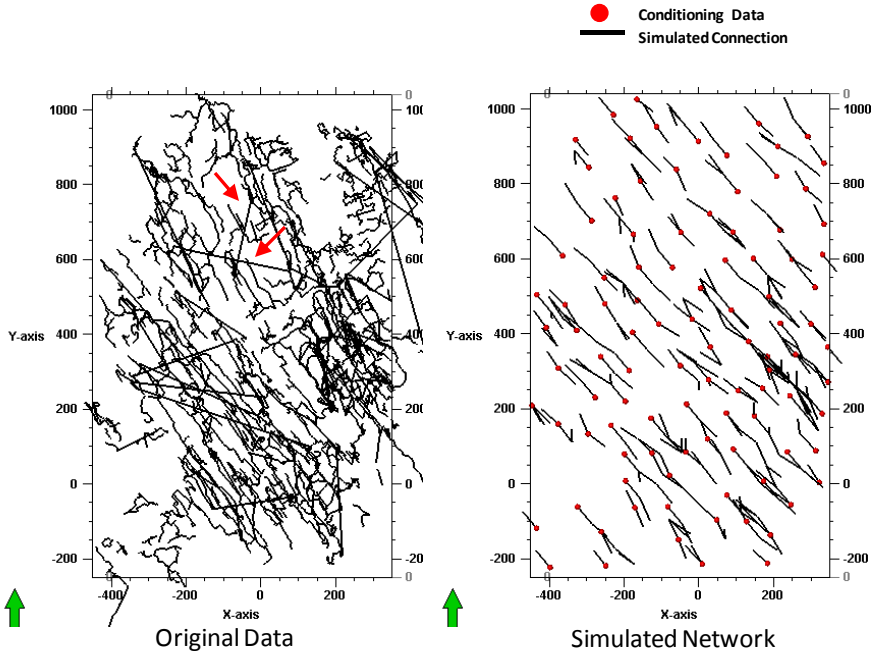


Figure 6.12- Pattern Simulation of Large Scale Features (Top View). Spurious passages in the original data (shown by red arrow) are artifacts of the cave network line plot. These connections are excluded in the analysis.

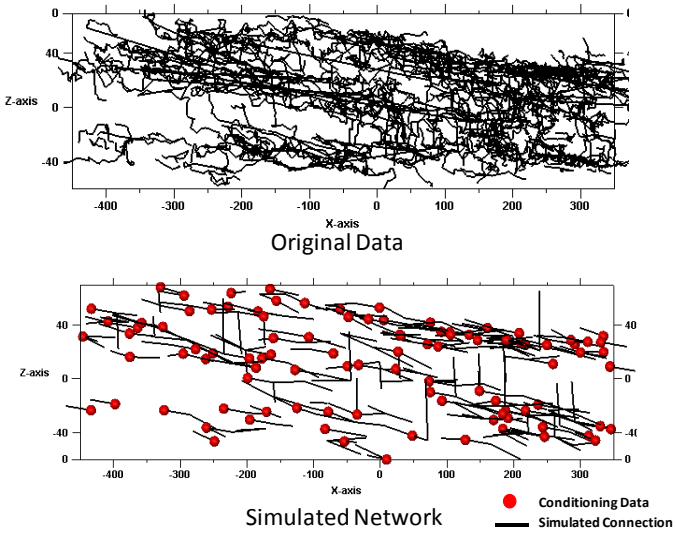


Figure 6.13- Pattern Simulation of Large Scale Features (Cross Section)

Mid scale features are simulated using Template 2 and corresponding pattern histogram (Figure 6.9). In Step 2, the entire set of points simulated in Step 1 is implemented as conditioning data. The tolerance window properties are not changed and mid-size cave passages are simulated following the main orientation of -40° azimuth direction with 5° dip angle. The simulated network at the end of Step 2 is given in Figures 6.14 and 6.15 where the results of the second step simulation are superimposed on the simulated large-scale patterns. In these figures, red points represent the original locations that are used for conditioning Step 1. The simulation is terminated at 3000 steps.

It is observed that mid scale passages are successfully simulated along the main trend in Step 2. Moreover, cave passage density is also reproduced by inheriting Region 2 quadrant analysis and it is observed that the servo mechanism implemented to maintain the cave density is working properly. The modeling mid-scale features results in very large and continuous cave passages that are obtained by growing patterns from previously

simulated features. Compared to the original cave map, the resultant network lacks any lateral connections between long passages and these structures are simulated in Step 3.

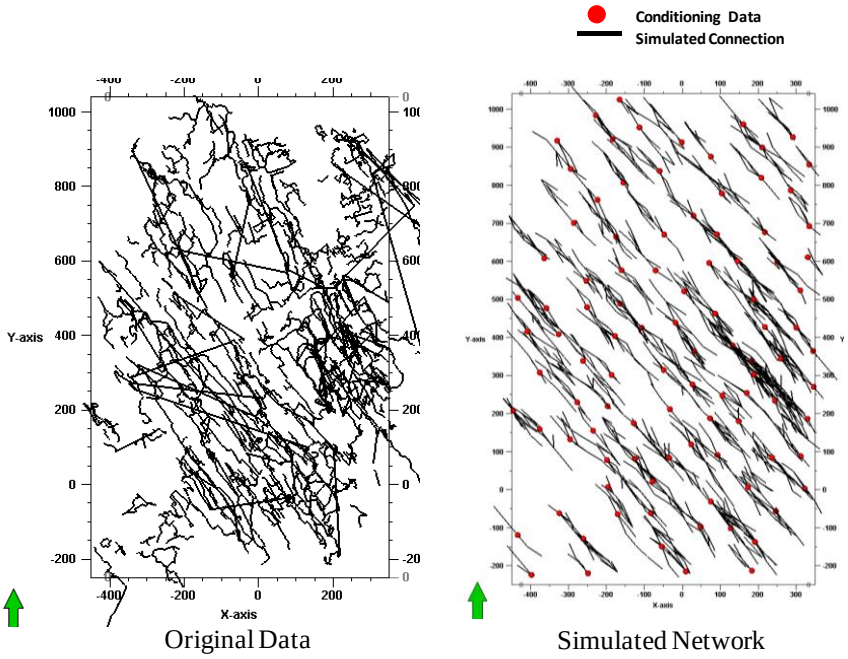


Figure 6.14- Pattern Simulation of Mid Scale Features (Top View). The original Wind Cave data and the simulated model are compared.

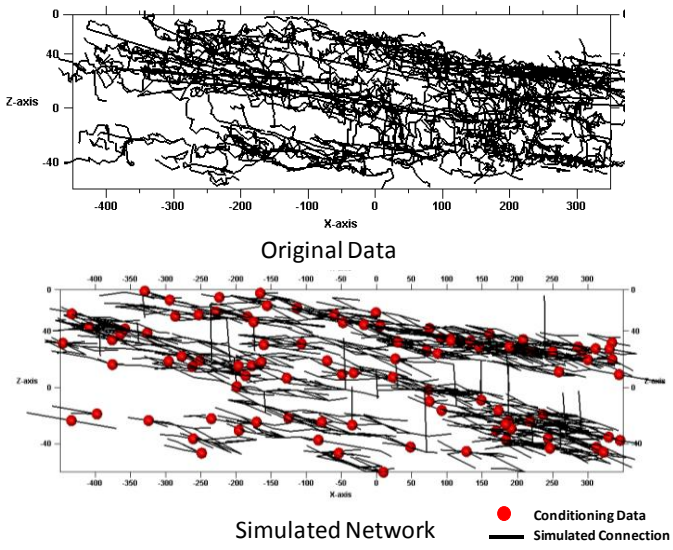
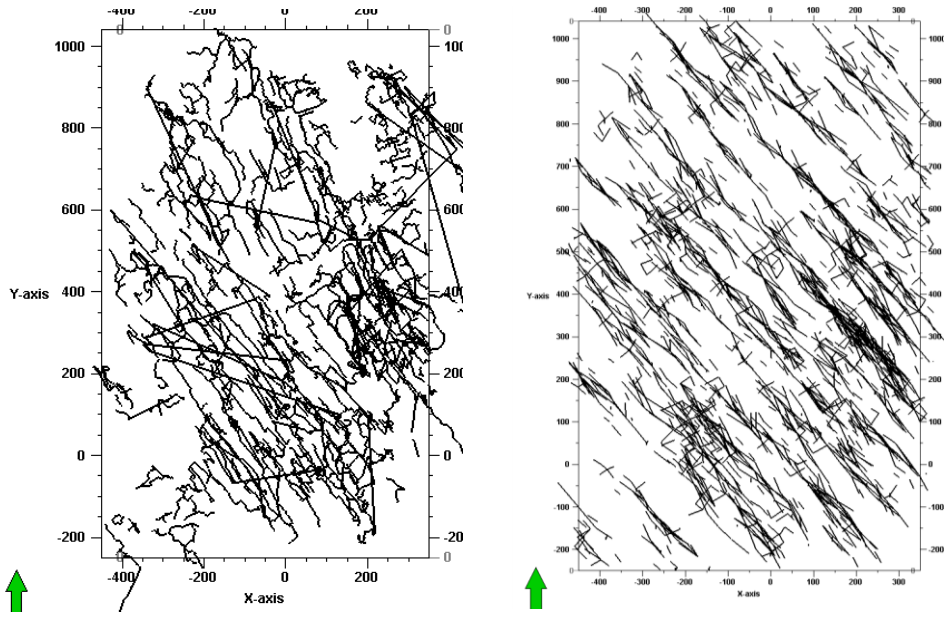


Figure 6.15- Pattern Simulation of Mid Scale Features (Cross Section)

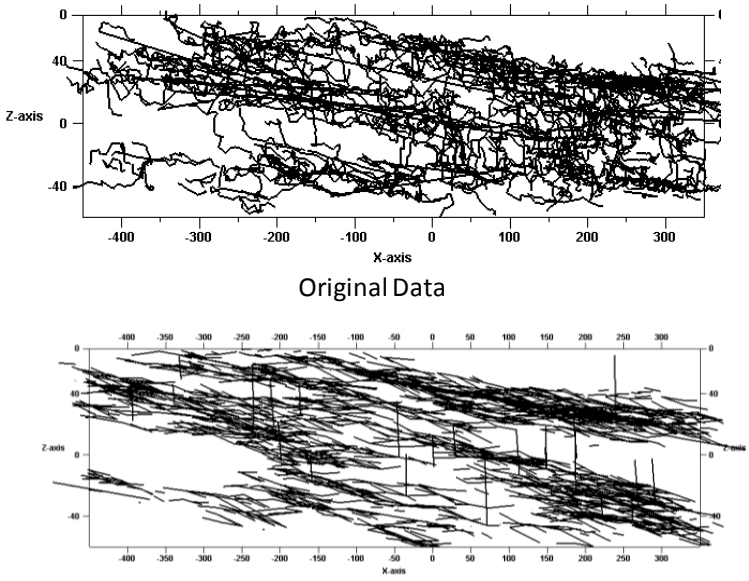
The last step of multi-grid pattern simulation of Region 2 aims to simulate lateral connections between the main cave passages. For this simulation using Template 3 is performed utilizing its pattern histogram (Figure 6.10). At the end of Step 2, the simulated point set has 1976 nodes. In order not to overwhelm the Step 3 simulation with too many conditioning data, 200 locations are randomly sampled from the simulated set with a minimum separation distance of 50 units and used for conditioning the Step 3 simulation. The number of conditioning points is selected based on user's preferences. For small-scale feature simulation, the number of conditioning data could be determined based on length distribution of cave passages if it is available. In case of available data, amount of small scale features can be used as a ratio of number of conditioning data to the total number of simulated points at the end of Step 2. Tolerance window properties are not changed for Step 3 and the simulation is terminated at the end of 1500 iteration steps.

Pattern simulation results of Step 3 are given in Figures 6.16 and 6.17 by superimposing them on the previously simulated network.



Original Data Simulated Network

Figure 6.16- Pattern Simulation Results for Region 2 (Top View)



Original Data

Simulated Network

Figure 6.17- Pattern Simulation Results for Region 2 (Cross Section)

In Step 3, small-scale structures are simulated and the final cave network is obtained. By visual comparison to the original data set, it can be concluded that the pattern simulation successfully reproduces the main structures of Region 2 caves such as large-scale and mid-scale continuous passages following -40° azimuth direction and, small-scale features along 50° azimuth angle that connect the main features. Besides pattern reproduction, the algorithm also honors conditioning locations and original distribution of cave passages by integrating a servo-system.

Another realization for Region 2 is obtained again using the multi-grid approach with the same templates. For Realization 2, tolerance window is slightly modified by increasing the dip tolerance to 20°. The simulation is also terminated after 1000 iteration steps in Step 1, 2000 iteration steps in Step 2 and 1500 iteration steps in Step 3. The simulated network is shown in Figures 6.18-6.23. It is observed that Realization 2 is also successful at reproducing cave network while honoring the major orientation, conditioning data and original cave passage distribution.

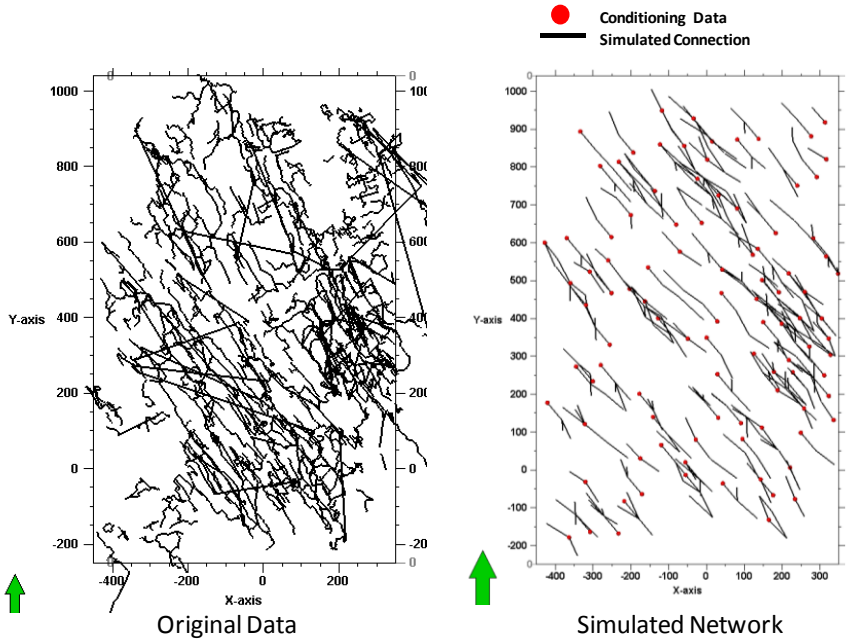


Figure 6.18- Pattern Simulation of Large Scale Features- Realization 2 (Top View)

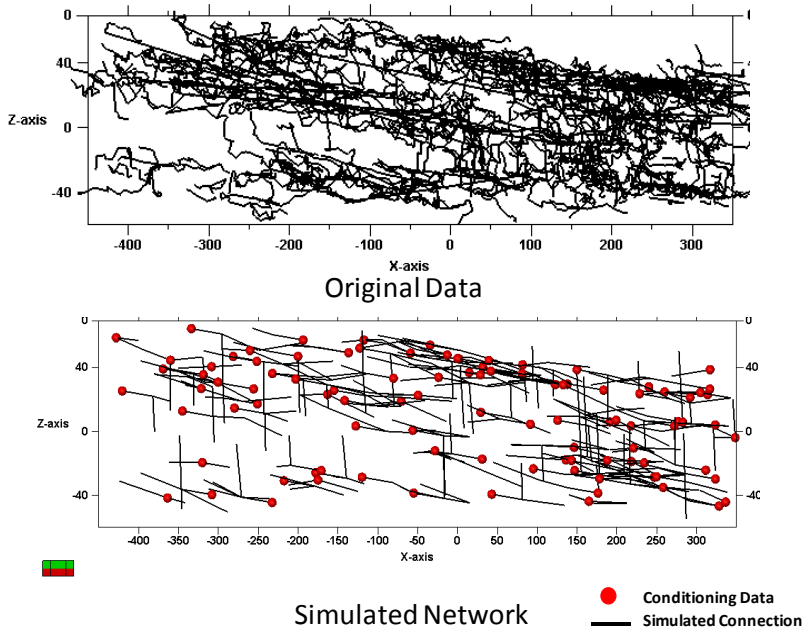


Figure 6.19- Pattern Simulation of Large Scale Features- Realization 2 (Cross Section)

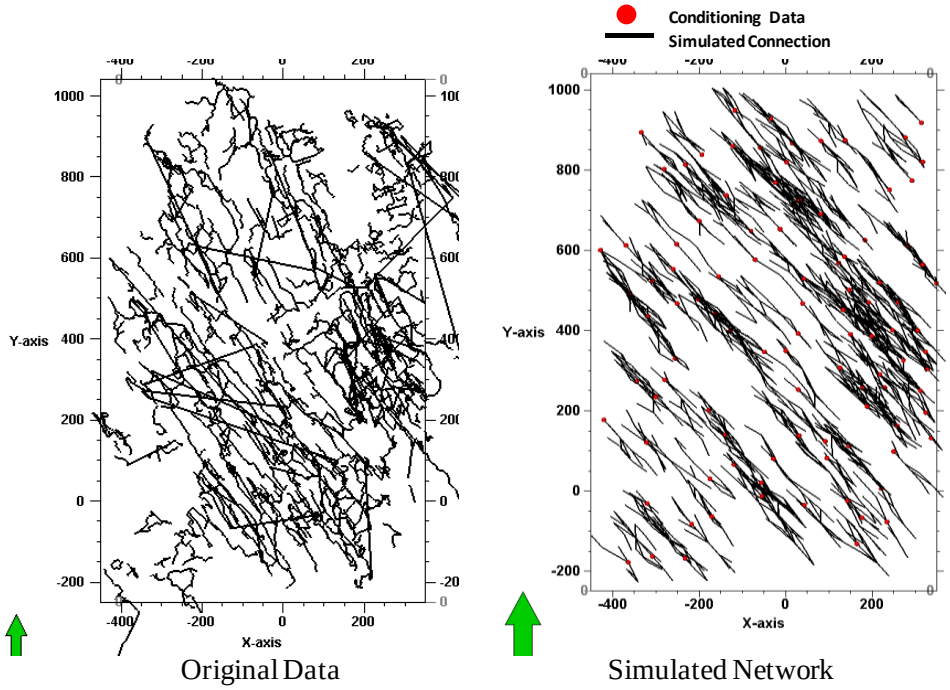


Figure 6.20- Pattern Simulation of Mid Scale Features- Realization 2 (Top View)

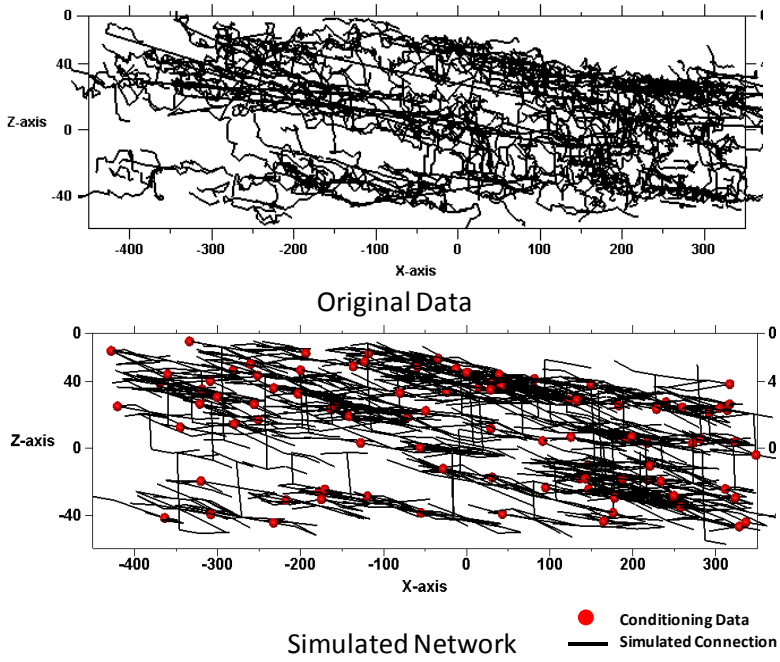
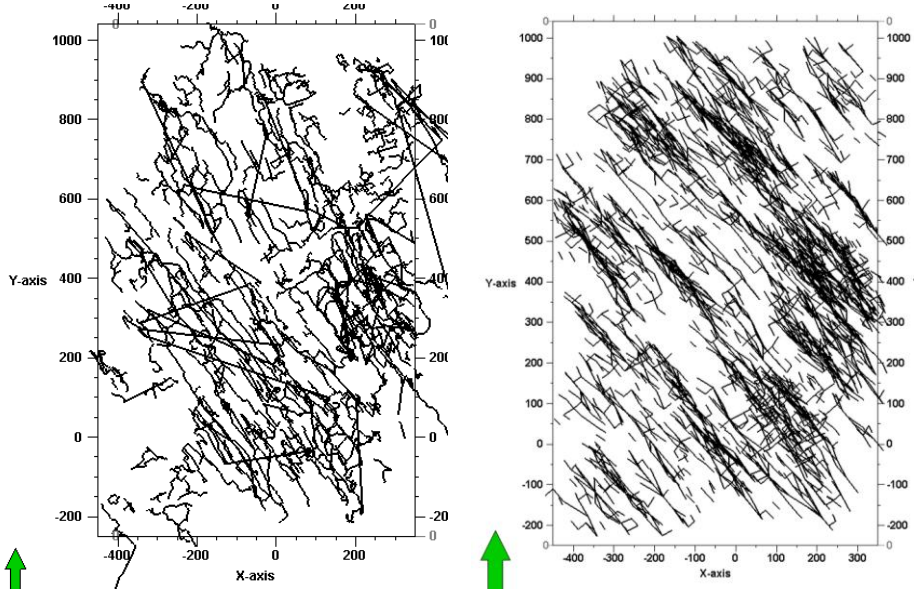


Figure 6.21- Pattern Simulation of Mid Scale Features- Realization 2 (Cross Section)



Original Data Simulated Network

Figure 6.22- Pattern Simulation Results for Region 2- Realization 2 (Top View)

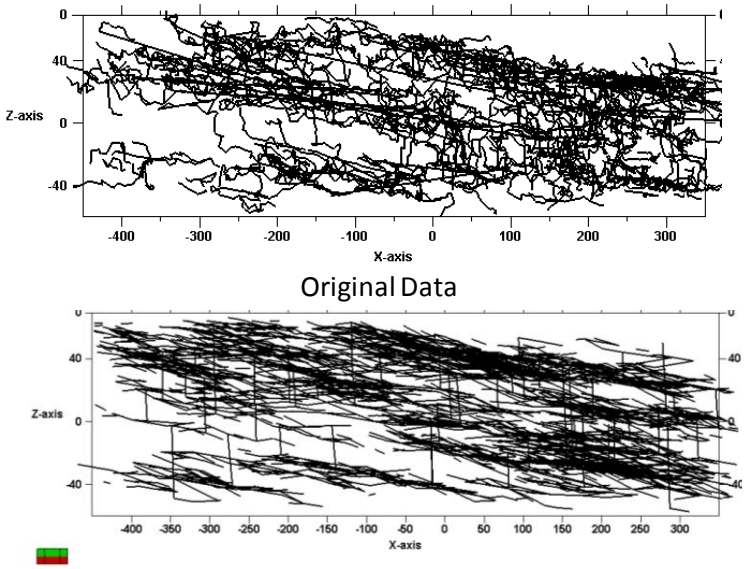


Figure 6.23- Pattern Simulation Results for Region 2- Realization 2 (Cross Section)

Because the realizations look very similar visually, it is important to compare them by computing MP-statistics. For this reason, the simulated point sets are scanned using Template 3 (Figure 6.10) with a tolerance window of 20° azimuth tolerance, 10° dip tolerance and a lag tolerance factor of 1/3. Then, the constructed pattern histograms are compared to the original one (Figure 6.24).

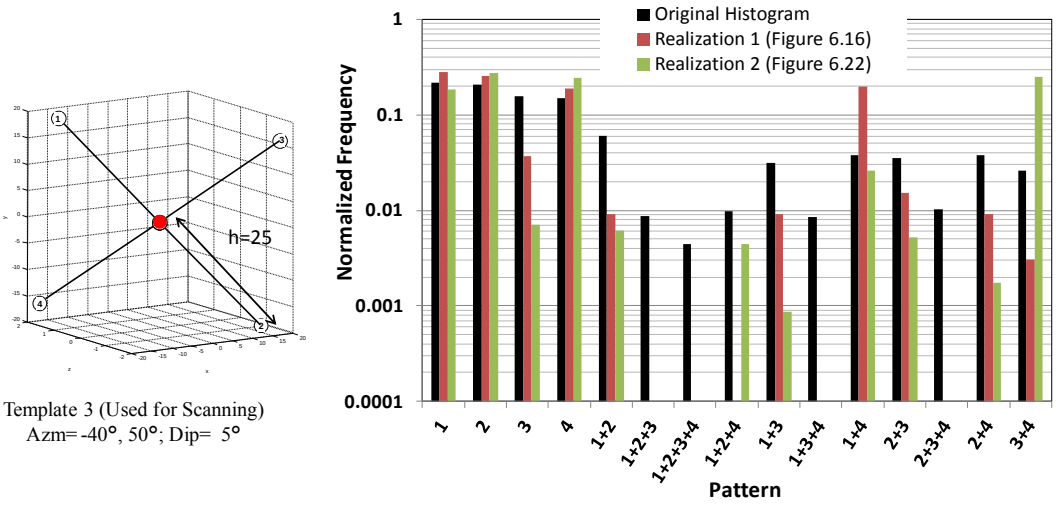


Figure 6.24- Pattern Histogram Comparison. Original histogram (given in black) is successfully reproduced in both realizations. Pattern histogram of Realization 1 (red) is slightly better than the one for Realization 2 (green).

Comparison shows that both the realizations successfully reproduce original pattern histogram. While the statistics for Realization 1 is closer to the original data, both realizations are fairly close to the target statistics indicating that the simulation procedure is robust regardless of small variations in user-defined parameters such as lag/azimuth tolerances, simulation steps etc. It is observed that some of the patterns are not present in the simulated histograms and these patterns can be reproduced by using different spatial templates, varying tolerance window size and increasing number of simulation iteration steps.

The non-gridded MPS analysis and pattern simulation algorithms successfully characterize and model cave networks that are described in the form of point-sets. Therefore, these tools will be used for modeling the paleokarst network of a West Texas reservoir. In Chapter 7, the application of the algorithms to the Yates Field is presented.

Chapter 7: Paleokarst Network Simulation of Yates Field

In this chapter, the non-gridded MPS algorithm is applied for the paleokarst network modeling of Yates field. First, an overview of Yates field and available field data are presented. Then, results obtained for paleokarst network simulation of Yates field are discussed. Finally, the paleokarst network is incorporated in a commercial black oil fluid flow simulator and the importance of performing the MPS simulation of complex cave connectivity for accurately predicting reservoir flow performance is demonstrated.

7.1 Geological Overview of Yates Field

Yates field is an important oil reservoir in a paleokarst setting located in Southeast tip of Central Basin platform in Texas with an asymmetric horseshoe shaped anticline (Figure 7.1). It was discovered in 1926 with original oil place of 5 billion bbl and the recovery factor is currently reported as 29%. There are more than 1,500 wells at 10-acre spacing in larger areas and the reservoir covers an area of approximately 26,000 acres (Campanella et al., 2000).

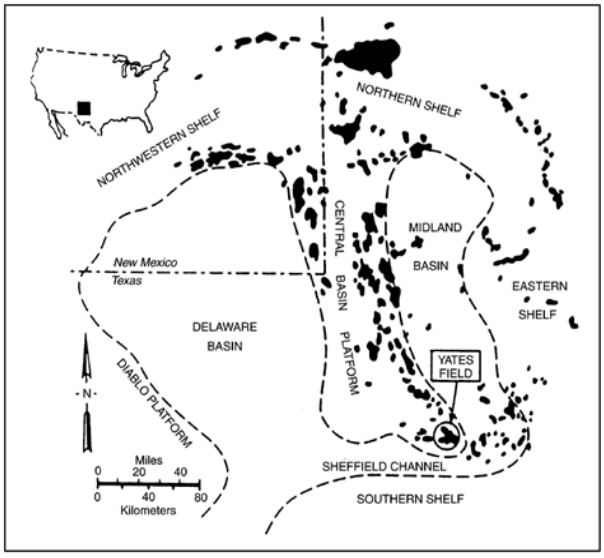


Figure 7.1- Location of Yates Field (Tinker et al., 1995)

The east and west sides of Yates field have different reservoir characteristics due to the cave system that is abundant in the eastern side. Thus, variations in petrophysical properties of the reservoir have significant impact on reservoir management decisions. Yates field had been producing by gravity drainage from 1929 until 1976. In 1976, waterflooding and polymer augmented waterflooding were started in the west side whereas and gas re-injection operations were initiated for pressure maintenance in the east side. From 1985 till 1991, CO_2 injection was conducted in the east side for tertiary recovery.

The main reservoir is Permian age San Andres Formation carbonate which is 96% dolomite in the eastern portions of the reservoir (Tinker et al., 1995). The field is relatively shallow at 1,500-2,000 ft depth. Although there are four producing zones, Seven Rivers, Queen, Grayburg and San Andres, the main reservoir is in the Permian age San Andres Formation (Figure 7.2).

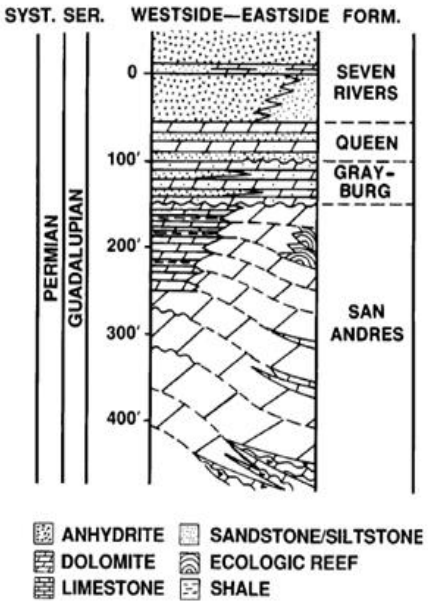


Figure 7.2- Stratigraphic Nomenclature for Yates Field (Tinker et al., 1995)

The Seven Rivers formation serves as a seal for the reservoir with an average thickness of 300-400 ft. The lower producing zone of the formation consists of sandstone with limited areal extent. On the other hand, the upper "carbonate" portion is composed of mainly anhydrite, gypsum and dolomite, and occasional sand and silt intervals. The main pore types identified in the core samples are micro-intercrystalline and intercrystalline pores. Remainder of the porosity comes from minor moldic, vuggy and fenestral pores suggesting a peritidal to supratidal depositional environment (Tinker and Mruk, 1995).

The Queen formation that underlies Seven Rivers has an average thickness of 45 ft throughout the field. The producing interval consists mainly of sand and silt. Based on the core interpretations, pore types in Queen formation are identified as micro-intercrystalline and intercrystalline and fenestral fabrics (Tinker and Mruk, 1995).

The Grayburg formation is the poorest reservoir rock with a thickness varying from 10 ft to 115 ft across the entire field. It is relatively thick at paleokarst depressions and extremely thin over San Andres karst topographic highs. Producing intervals in Grayburg formation are mainly dolomite and it has silt as the clastic component. The dominant pore types observed in the core samples are micro-intercrystalline, fenestral and vuggy pores.

The main producing zone in Yates field is the San Andres Formation that is almost 95% dolomite in the east side of the field. Reservoir rock quality degrades with increasing shale content towards the western regions. San Andres formation has a thickness of up to 750 ft and it is composed of three principal lithofacies; (1) lagoonal shales, dolomitic mudstone and wackestone in the west side, (2) higher porosity dolomitic packstone and grainstone shoal complexes of deep west side, upper east side of the field and (3) low porosity dolomitic packstone in the eastern slope and deep ramp

(Tinker and Mruk, 1995). Karst system in Yates field is formed in porous and permeable limestone of San Andres formation with extensive dissolution along joints and the early karstic features are altered by dolomitization (Tinker et al., 1995).

Stratigraphic cross section and major depositional cycles in Yates field are given in Figure 7.3. Highest permeability and karstification are mostly recognized in carbonate shoal packstone and grainstone formations of the Yates field to the east (Tinker and Mruk, 1995).

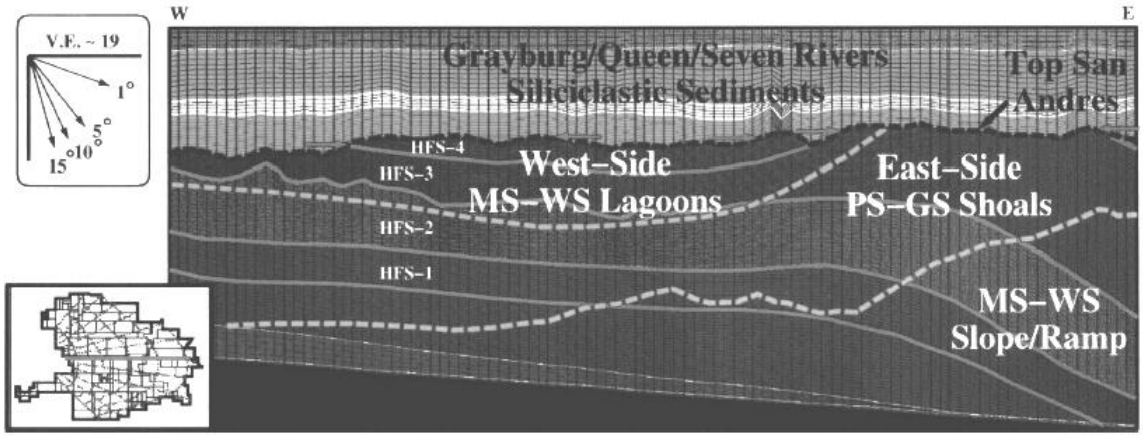


Figure 7.3- Cross Section and Depositional Cycles of Yates Field (Tinker and Mruk, 1995)

Several studies were conducted for detailed description of the formation and identification of diagenetic events in the Yates field (Spencer and Warren, 1986; Craig et al., 1986; Tinker et al., 1995). Multiple unconformity surfaces were observed and extensive karstification was identified. According to Tinker et al. (1995), there is a strong relation between cave distribution and sequence stratigraphy. By using core and well log data, they identified four major cycles of aggradation and progradation from west to east and each cycle was followed by surface exposure. During the subaerial exposure, cave lenses are formed by meteoric processes in mixing zones and since the reservoir is relatively shallow, most of the caves remain open (Tinker et al., 1995). Therefore,

subaerial exposure and karstification resulted in a network-maze cave pattern that is controlled by pre-existing joints in the field.

Cavernous porosity and open caves were identified below the unconformity surfaces. The paleokarst system is dominant in the east side of the reservoir and several evidences are observed in wells such as numerous bit drops, well log anomalies and recognized paleokarst facies in core samples. Previous studies on recognition of cave network in Yates field are based on bit drops, well logs and core descriptions. Some of them are presented in the following section.

7.2 CURRENT STUDIES ON PALEOKARST SYSTEM IN YATES FIELD

The field operator identified a maze type paleokarst network in 1996. Their interpretation of the cave map was based on delineation of bit drop data and well log information (Figure 7.4).

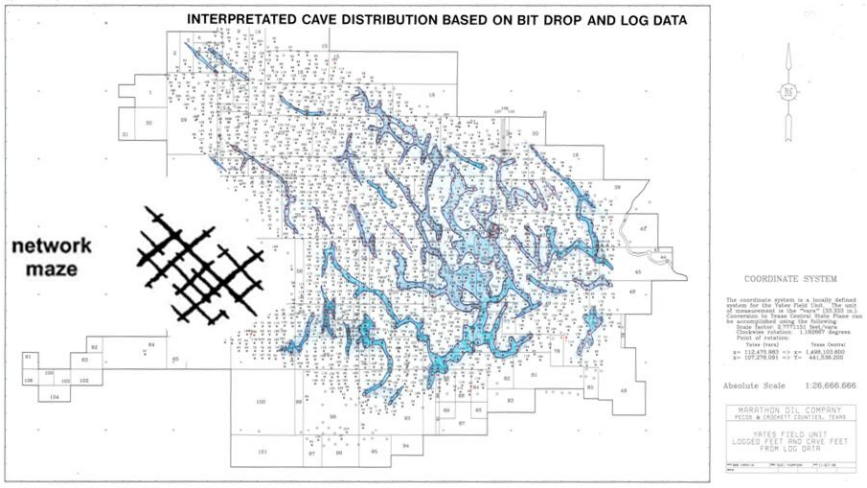


Figure 7.4- Cave Distribution Map based on Bit Drop and Log Data (Behnken and White, 2007)

The company proposed that the cave structure is a maze network formed by the extensive dissolution of soluble rock along intersecting fissures (Palmer, 1991). According to Palmer (1991), maze networks are formed by (1) surface runoff caused by

sinking streams or rivers during floods, (2) diffuse recharge through fractures or inter-granular pore space, (3) dissolution of the porous media caused by mixing of waters with different chemistry or by cooling of thermal water. Palmer's definition of maze network and the proposed cave map (Figure 7.4) are in an agreement. However, the cave map is constructed by quick interpretation of bit drop and well log data and it does not reflect the actual connectivity of the network system. Moreover, the wide sections on the cave map do not represent the true dimensions of the subsurface paleokarst structures.

Tinker et al. (1995) has conducted an extensive study of the sequence stratigraphy of San Andres formation using a core database of 23,000 ft from 118 wells and log data of 1800 wells. They proposed a technique to identify cave zones by using gamma ray, bulk density and caliper logs. According to Tinker et al. (1995), a cave is determined if GR is less than 40 API, bulk density is less than 2g/cc and an excursion from baseline in the caliper is present. By using the proposed criteria, they determined 1050 caves in Yates field. Tinker et al. (1995) integrated core interpretations and identified cave zones to obtain a distribution of cave structures in the field. They calculated total cave distribution in each cycle of San Andres formation by defining the "logged foot" as one vertical foot of wellbore with log data and "cave foot" as one vertical foot of cave. They used these definitions to compute the cave feet per 1000 logged feet that is presented as a function of depth with respect to Seven River M datum in Figure 7.5.

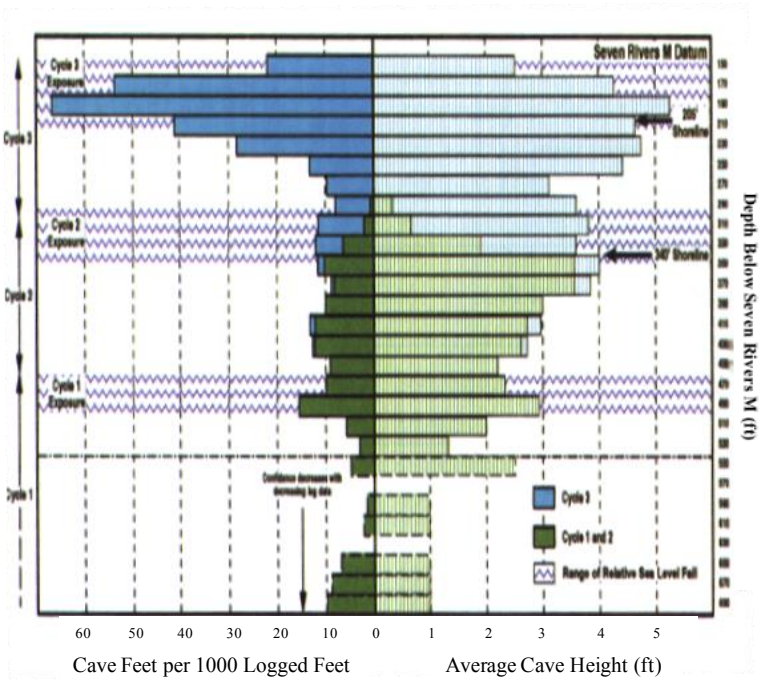


Figure 7.5- Cave Feet and Cave Height as a function of depth below the Seven Rivers M Datum (Tinker et al., 1995)

Tinker et al. (1995) showed that the majority of the cave structures are in the upper intervals of San Andres formation and caves are abundant in Cycle 3. On the other hand, Cycles 1 and 2 have few caves while Cycle 4 is completely eroded in the eastern part of the Yates field.

According to Tinker et al. (1995), increase in cave heights toward the top of each cycle proves that San Andres formation was exposed to the surface at the end each deposition cycle and meteoric diagenesis was the cave forming mechanism. Craig (1988) suggested that the karst system in Yates field was formed by Island Karst Model in which dissolution is because of mixing of sea water and meteoric waters. Tinker et al. (1995) has also applied Island Karst Model to cave formation of Yates field (Figure 7.6). They supported the proposed model by investigating the cave distribution in Cycle 3. They identified that the highest cave feet and largest caves are located at the mixing zone of

vadose and phreatic fresh waters (Figure 7.6) and similar conclusions can be obtained for Cycles 1 and 2. Thus, the main driving mechanism for dissolution in San Andres formation is the mixing of waters with different chemistry.

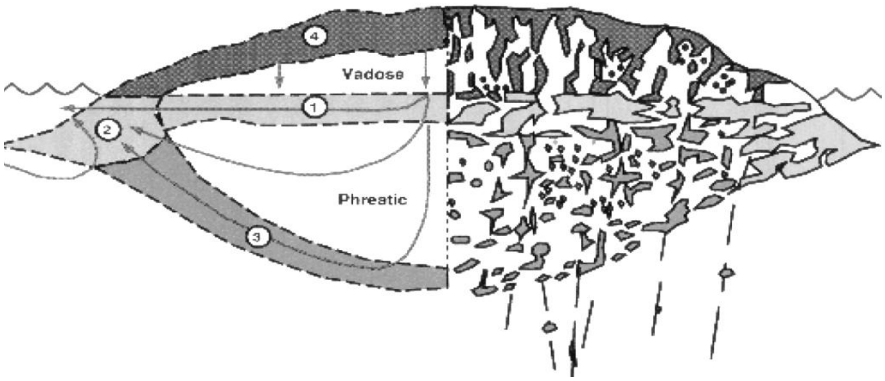


Figure 7.6- Island Karst Model Applied to Yates Field; cave formation in sites 1, 2 and 3 due to mixing, in site 4 associated with soil processes (Tinker et al., 1995). The main driving mechanism for dissolution in San Andres formation is the mixing of waters with different chemistry at Region 2.

Tinker and Mruk (1995) have also carried out a comprehensive study of core data from the Yates field. They examined 118 cores and presented rock-fabric descriptions, porosity, permeability and fracture analysis, pore type and calcite distributions, and used that information to develop a static 3D reservoir model. According to Tinker and Mruk (1995), the current cave distribution and karst lineaments are mostly affected by the early regional fracture and joint system. Therefore, final distribution of the paleokarst system follows the regional lineament trend of N50°W and N40°E.

These previous studies show that there is a significant variation in petrophysical and rock fabric properties from east to west of the reservoir due to the abundance of cave structures in the eastern Yates. Thus, Button and Peterson proposed “a line of demarcation” to divide paleokarst bearing east zone from west side of Yates field for

better reservoir flow performance analysis and better reservoir management decisions (as cited in Cheng and Kwan, 2012) (Figure 7.7).

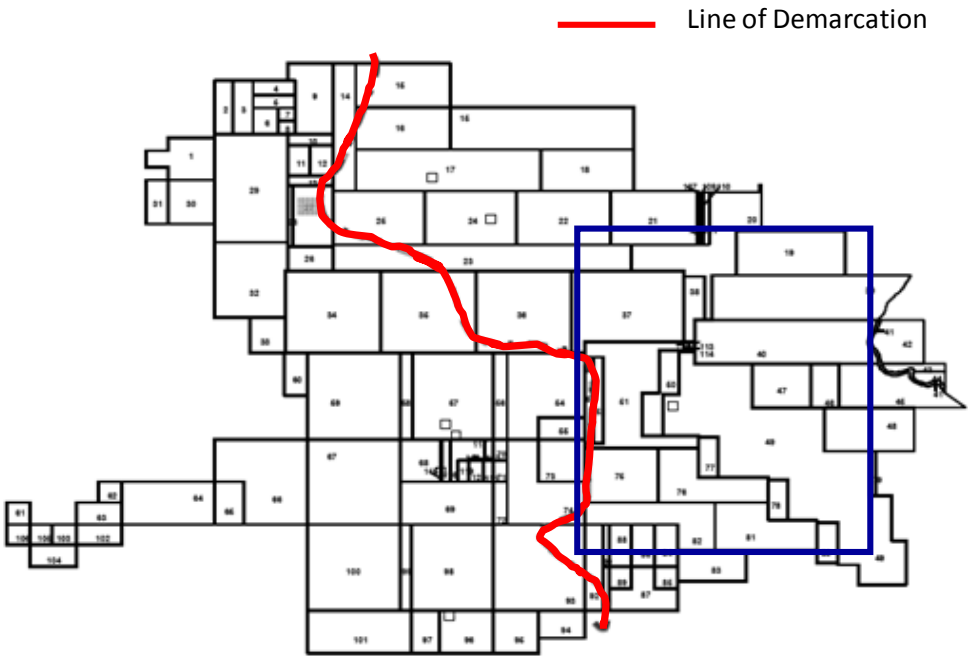


Figure 7.7- Line of Demarcation for Yates Field (adapted from Cheng and Kwan, 2012). Black lines show tract boundaries. The area inside the blue rectangle is used in this study.

Since the cave network is abundant in the east side of Yates field, a small area is selected from that region for further analysis of paleokarst identification, cave network modeling and flow simulation studies.

7.3 Identification of Paleokarst Facies Using Field Data

In this study, we focused on a small region selected from the eastern Yates field. This selection was based on the availability of field data, cave map based on bit drops (Figure 7.4) and cave zone thickness analysis of Tinker et al. (1995). Well logs and core data are available for wells inside the region bounded by the blue boundary in Figure 7.8.

For detailed analysis, a small region is selected and it is shown by the red area in Figure 7.8.

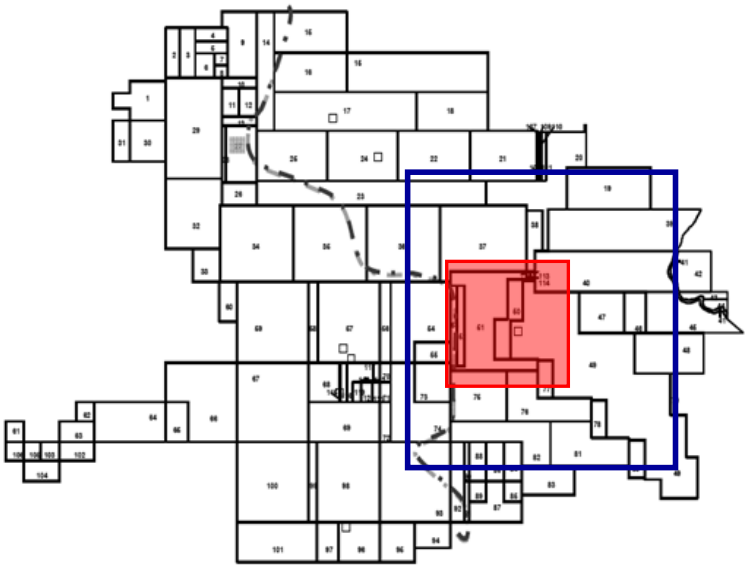


Figure 7.8- Region selected for analysis. Well data is available for the region inside the blue boundary. Further analysis is conducted for the small region inside the red area.

The small region has more than 200 wells including, vertical and horizontal production wells, gas and water injection wells and abandoned vertical wells. Well logs (caliper, gamma ray and bulk density), core information and oil production profiles are available for some of the wells. In this study, we focused on vertical production wells during the primary depletion period (1926-1976). Initial analysis for identifying cave facies is performed by using well log data. Then, core information is integrated to validate the log analysis. At the end of this process, an indicator data set for cave facies as well as the distribution of gross thickness of cave facies is obtained for the small region (Figure 7.8). The indicator data set is used for the cave center line (MPS) simulation.

7.3.1 Cave Facies Identification Using Well Log Data

Tinker et al. (1995) previously proposed a criterion for recognizing cave zone using well log information. Their identification was mostly based on gamma ray, density logs and caliper excursions. According to Tinker et al. (1995), cave zone is described by GR less than 40 API and bulk density less than 2 g/cc with an excursion from baseline in the caliper. In this research, an alternate set of criteria for identification of cave facies was developed and compared against the intervals identified as cave facies using Tinker's criteria.

First, Tinker's criteria are applied for the small region and well logs are examined in detail for identification of possible cave facies (Figure 7.9). In this figure, purple facies indicate cave zone while green intervals are non-cave facies. It is observed that these criteria have some drawbacks. In some cases, cave intervals are reported without any caliper excursion (Figure 7.9, middle figure). These problems indicate that recognition of cave facies with Tinker's criteria mostly depends on GR and bulk density logs ignoring the evidence presented by caliper deviation. Also, use of a low bulk density cut off might cause underestimation of cave zone thicknesses. Therefore, well logs for the entire small region are evaluated by modifying Tinker's criteria.

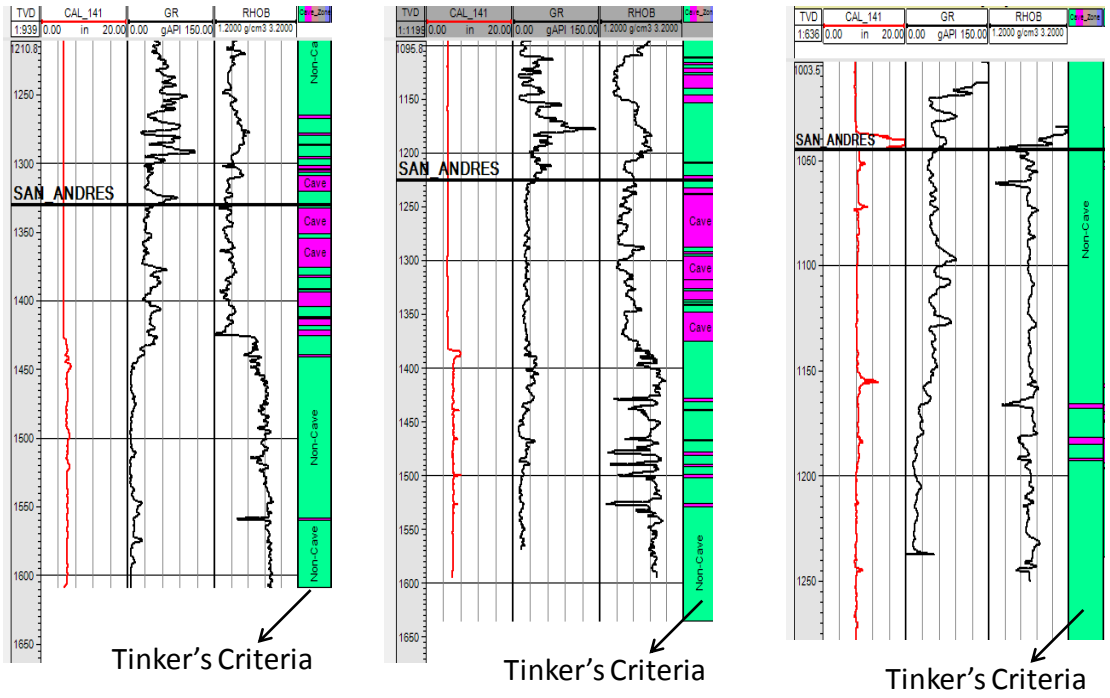


Figure 7.9- Comparison of Identified Cave Zone Intervals -1. Pink indicates cave facies and green intervals are non-cave facies.

The modified criterion utilizes a qualitative measure of caliper response as well as GR and bulk density values to identify cave facies. First, the bulk density cut off is increased to 2.3 g/cc based on an evaluation of Tinker's criteria. Then, a qualitative measure of caliper deviation is defined for cave facies identification. If GR and bulk density values are below the cut off and there is a minimum caliper deviation of 1.5 in from a base line, that specific interval is considered as a cave zone. These modifications are used to calculate cave zone thicknesses below the San Andres formation in the small region and the results are compared to the ones obtained by Tinker's criteria (Figure 7.10). It is observed that in general, the identified cave zone intervals are in agreement with the ones by Tinker's criteria (Figure 7.9). Moreover, the modified criteria perform

better in some intervals where the caliper deviation is insignificant and Tinker's criteria identifies that particular interval as cave zone (Figure 7.10) which is counter-intuitive.

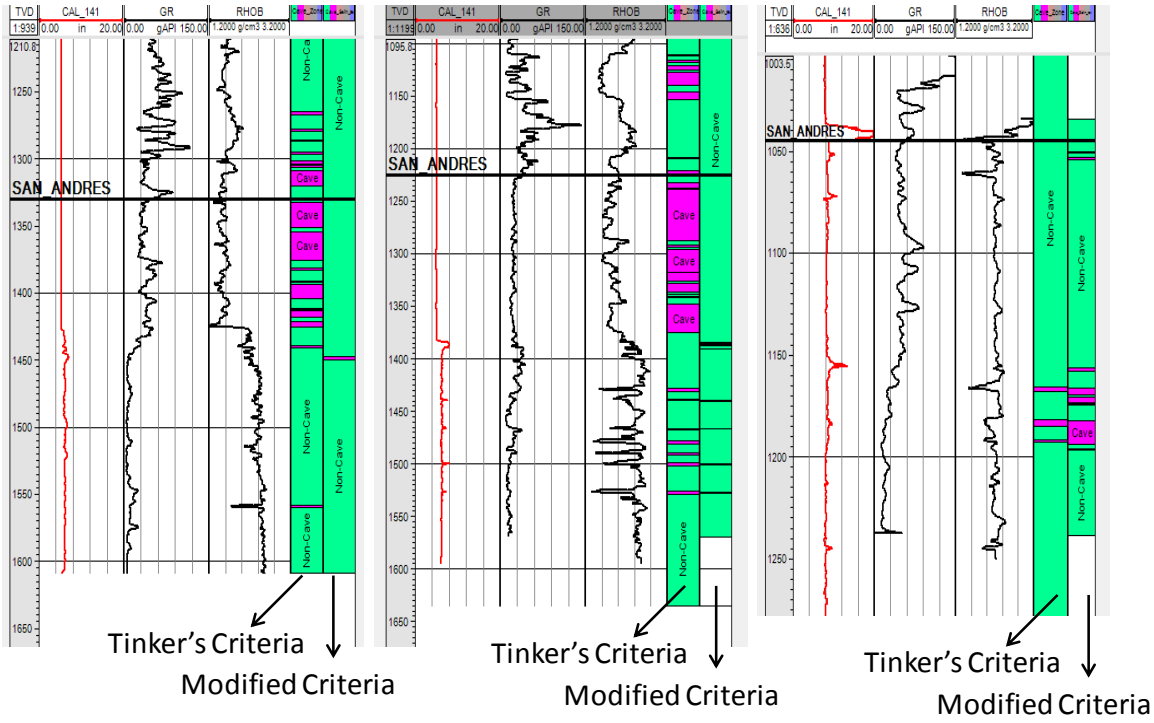


Figure 7.10- Comparison of Identified Cave Zone Intervals -1. Purple indicates cave facies and green intervals are non-cave facies identified using Tinker's criteria compared against that using the modified criteria.

Although the modified criteria perform reasonably well, threshold values for GR and bulk density should be verified. A GR threshold value of 40 API is reasonable since there exist shale layers in San Andres formation that are recognized by GR values greater than 40 API (Tinker and Mruk, 1995). Thus, it is concluded that GR criterion is valid. On the other hand, well log and core analysis data are examined in detail to validate the bulk density threshold.

First, cross-plots and histograms for GR and bulk density logs are constructed to understand whether there is a correlation between them. In this analysis, shale layers

(GR>40 API) are excluded for accurate predictions in dolomite interval and plots are obtained for 6 wells. It is observed that there is no significant correlation between bulk density and GR values (Figures 7.11-7.12).

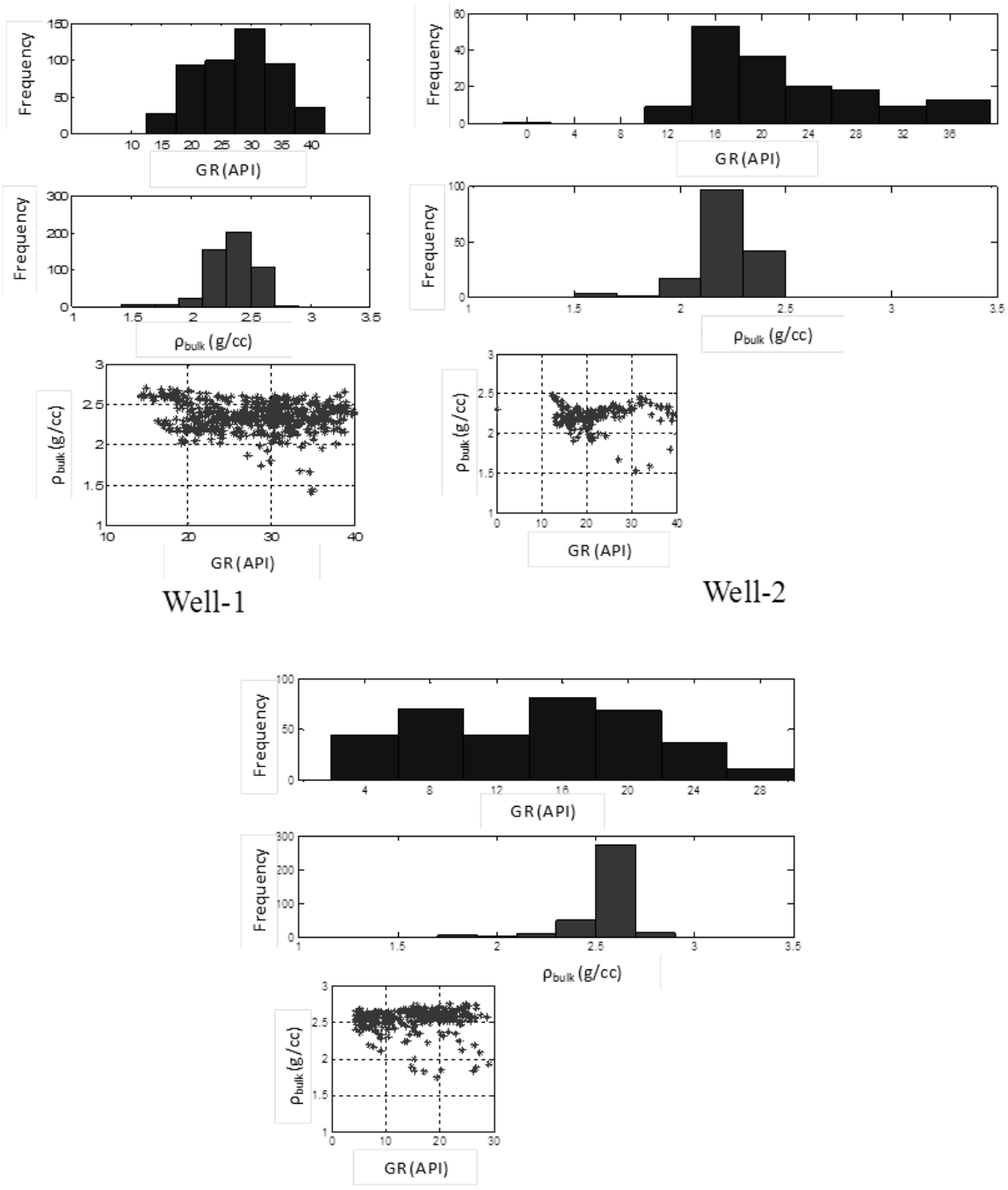


Figure 7.11- Well Log Histograms and Cross-plots -1

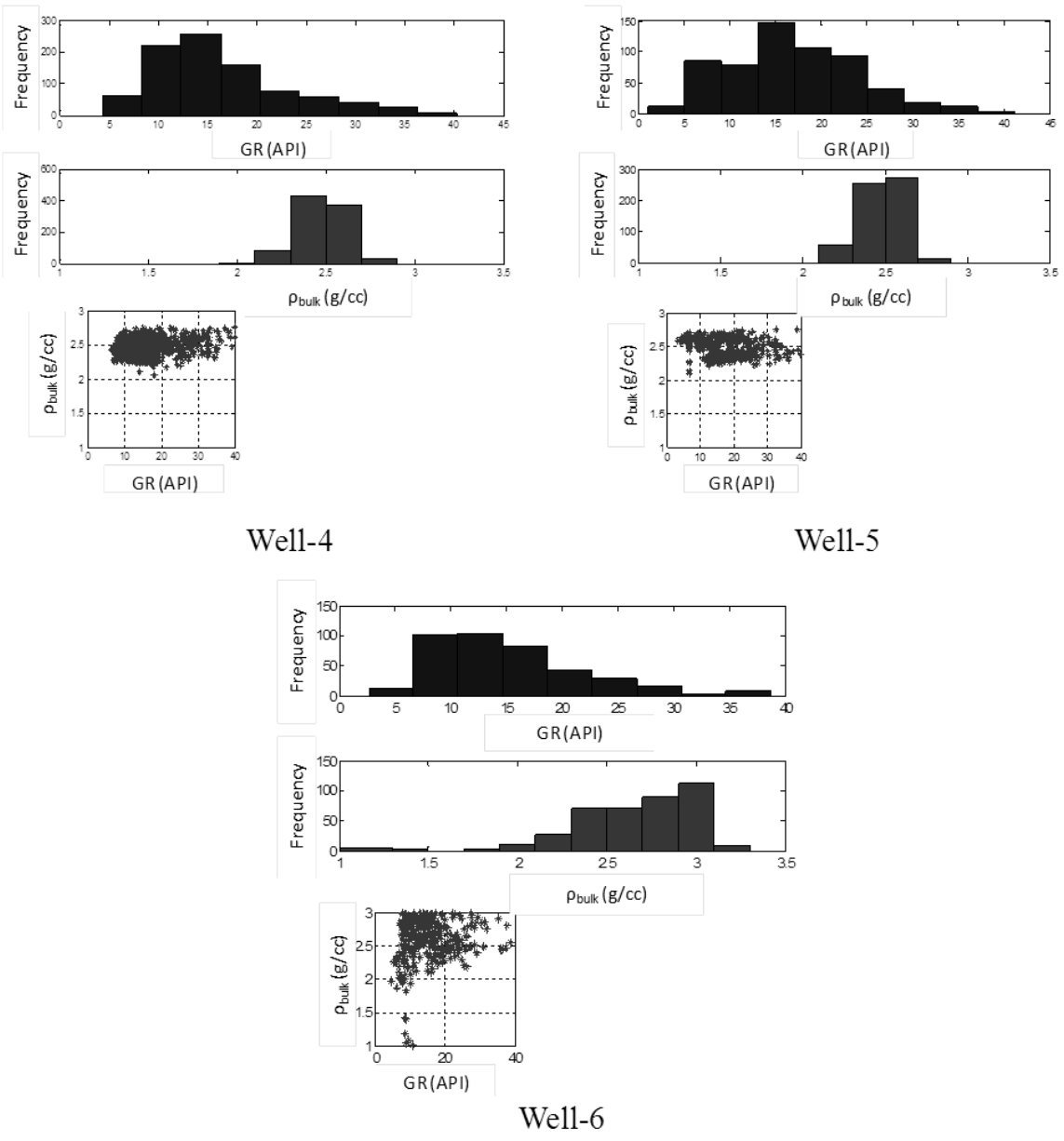


Figure 7.12- Well Log Histograms and Cross-plots -2

To identify an indicator for cave facies intervals, k-means clustering analysis (Kanungo et al., 2002) is performed on the cross plot between bulk density and GR. In Figures 7.13-15, k-means clusters and corresponding cluster depths are illustrated for Wells 1, 2 and 6 respectively. Bulk density values are divided into an optimum number of

3 clusters based on the variations with GR. Then, cluster variation of bulk density with respect to depth is examined (Figures 7.13-7.15). It is determined that density values of certain clusters fall into predicted cave zones (Figure 7.13 mid section, Figure 7.14 upper section and Figure 7.15). Therefore, the cluster means corresponding to these sections are examined (cluster mean values are shown in the figures). It is identified that clusters in agreement with the predicted cave intervals generally have k-mean bulk density greater than 2 g/cc (Table 7.1). Thus, it is concluded that the modified criteria with bulk density threshold value of 2.3 g/cc may successfully predict possible cave intervals.

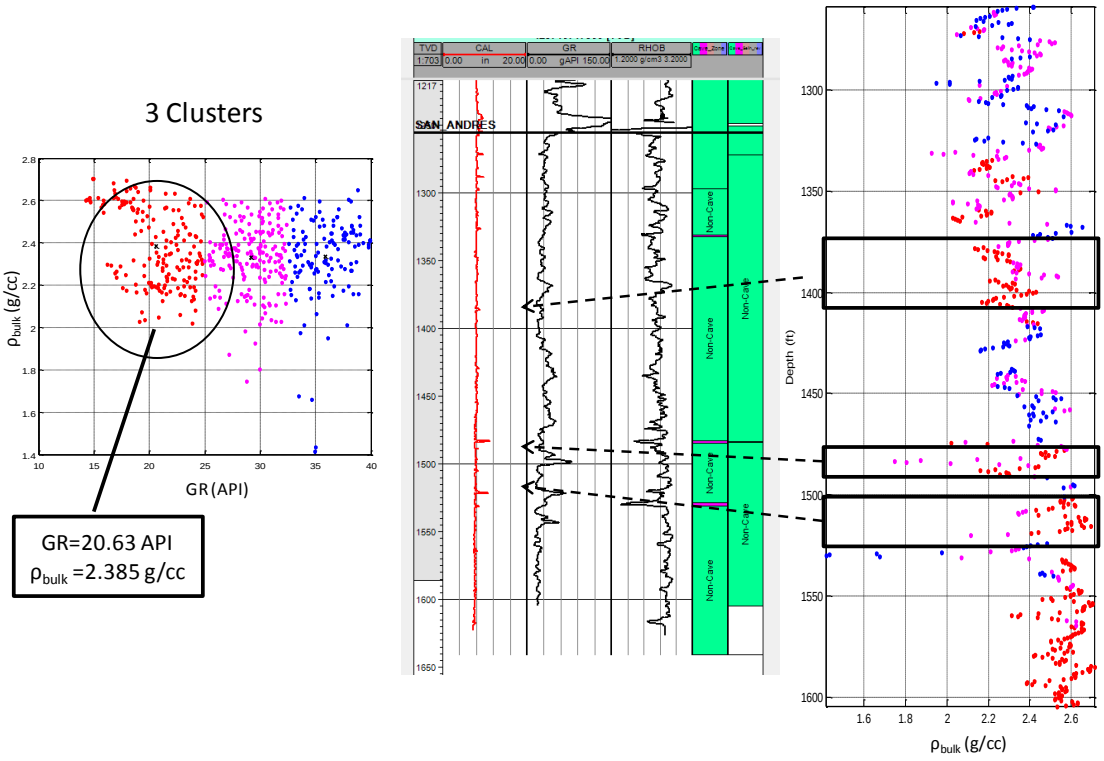


Figure 7.13- k-means cluster analysis of Well 1. The red cluster is generally in agreement with cave facies and caliper anomalies. Boxes and arrows show the depths corresponding to the red cluster.

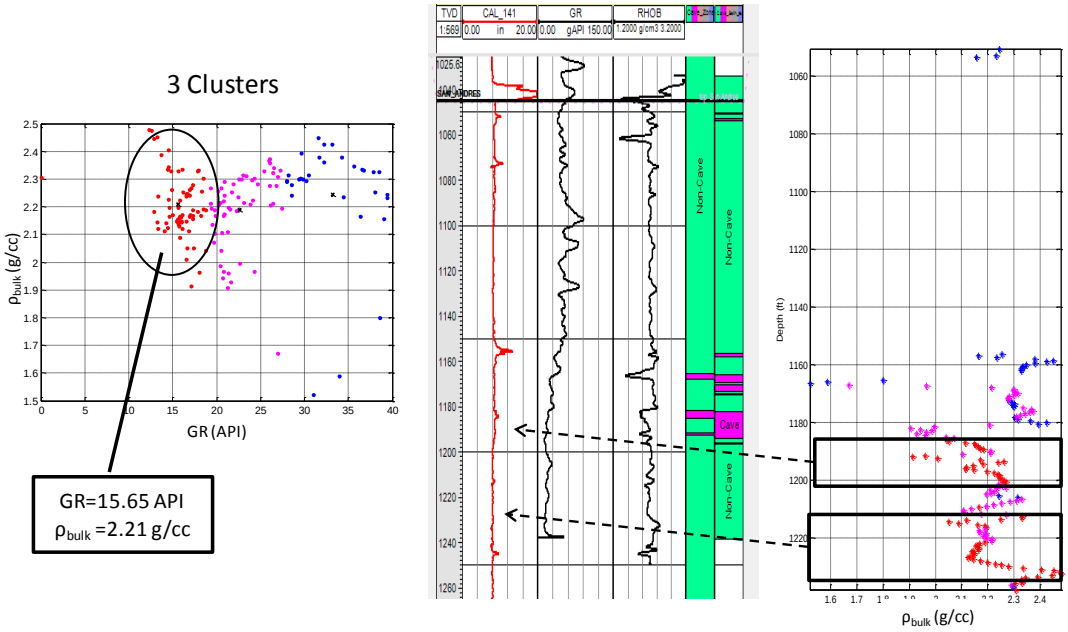


Figure 7.14- k-means Clustering of Well 2. Red cluster is in fair agreement with cave zones and caliper variations. Boxes and arrows show the depths corresponding to the red cluster.

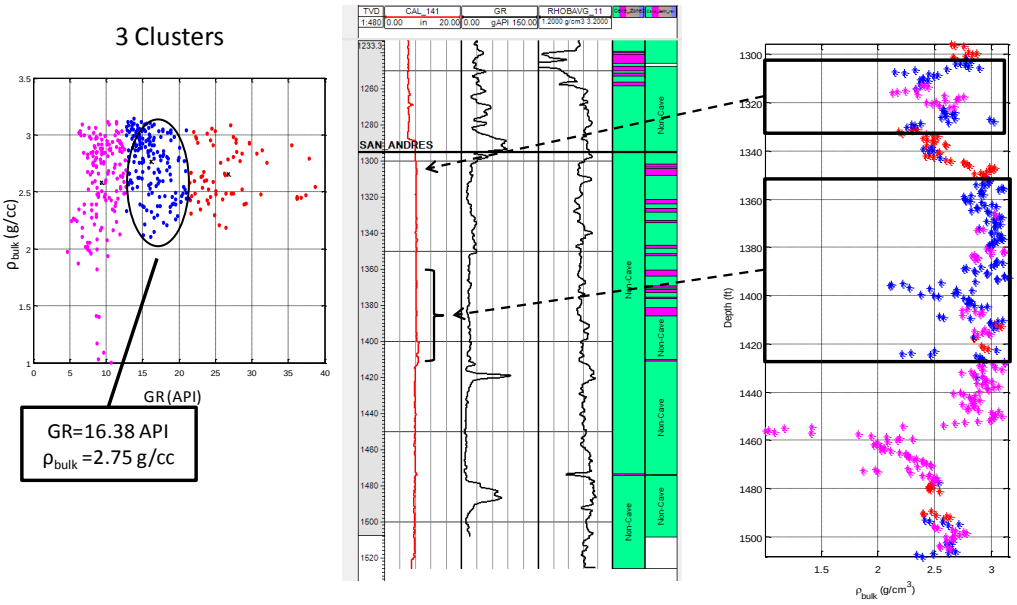


Figure 7.15- k-means Clustering of Well 6. Blue cluster is generally in agreement with cave zone and caliper variations from the base line of 7.5". Boxes and arrows show the depths corresponding to the blue cluster.

Table 7.1- Cluster Mean Properties and Predicted Cave Zone Gross Thicknesses. Caliper deviation > 1.5 in is explicitly used to define the cave zone.

Well Number	GR (API)	ρ_{bulk} (g/cc)	Cave Zone Gross Thickness by Modified Criteria (ft)
1	20.63	2.38	0.5
2	15.65	2.21	22.5
3	15.81	2.55	0.5
4	10.14	2.46	3.5
5	16.43	2.44	27.5
6	16.38	2.75	26.5

7.3.2 Verification of Cave Intervals Using Core Data

Cave identification mostly depends on well log data and it is essential to verify accuracy of the predicted zones. Since, the recognized cave intervals will be used for calculation of cave zone thickness distribution and conditioning cave network simulation, results obtained with the modified criteria are compared to core information for validation purposes. Core analyses of 9 wells are examined in detail and compared to the predicted cave/non-cave intervals using log values.

First, predicted cave intervals and core descriptions are analyzed for Well A and the cored interval is below the identified cave zone (Figure 7.16). As given the core reports, anomalies in caliper and density logs are in 1632-1693 ft interval. The presence of these cave indicators is consistent with the cave zone predicted by the modified criteria (Figure 7.16).

Analysis for Well B is conducted specifically for the predicted cave zone at 1264-1266 ft and this interval has core descriptions (Figure 7.16). Core permeability, porosity and texture descriptions are examined. Core samples in 1264-1267 ft interval are described as packstone-grainstone with high permeability (113-346 md) and high

porosity (0.22-0.29) indicating presence of cave facies. Also, caliper and density log anomalies in this particular interval are reported as cave zone indicators. Thus, the predicted cave facies zone is verified by the core data.

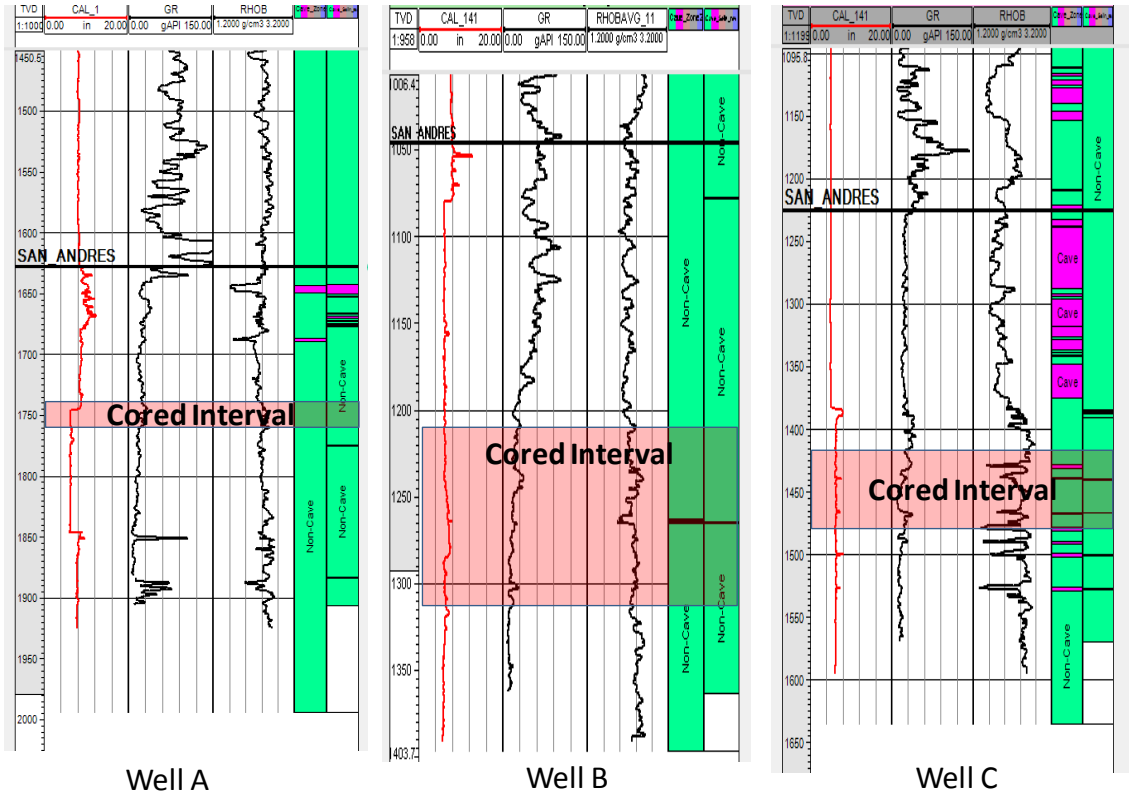


Figure 7.16- Well Section and Cored Intervals for three wells in the Yates field.

Similar analyses are conducted for at Well C completed in the predicted cave facies zone. Caliper and density log anomalies above the cored interval (1384-1400 ft) are noted as possible cave indicators and it is consistent with the cave zone predicted by the modified criteria (Figure 7.16). Cores in the 1436-1438 ft interval that is recognized as cave facies zone are also analyzed. In this interval, core samples are described as packstone with high permeability (186-231 md) and high porosity (0.27-0.30), and the presence of dissolution features are reported. These results support the presence of cave

facies in 1436-1438 ft interval that is predicted by the modified criteria. In addition, cave facies are also identified at a depth of 1466 ft and the corresponding core data are examined. Although there is no core recovery in the 1465-1469 ft interval, two samples taken below and above are considered. The first sample is from 1463 ft and it is described as packstone with high permeability (235 md) and high porosity (0.287). The second sample is a wackestone-packstone recovered at 1470 ft depth. This sample has relatively lower permeability of 24 md and porosity of 0.17. The sample is highly altered; features like vugs and dissolution channels are identified (Figure 7.17). Analysis of these core samples indicates that there is a high possibility of having a cave zone at the particular depth of 1466 ft.

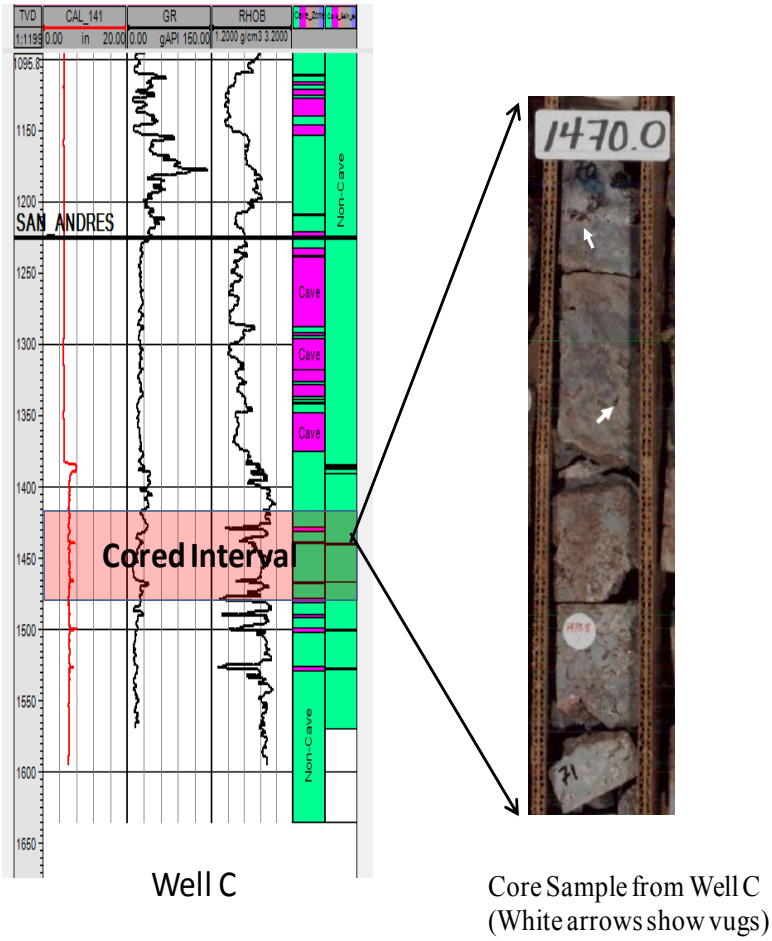


Figure 7.17- Core Samples from Well C. White arrows show vugs. Core image is provided by KinderMorgan.

Further core analyses and comparisons are conducted for the remaining wells. For Well D shown in Figure 7.18, there is no cave interval identified by the modified criteria (especially in the upper sections, due to lack of bulk density data for this particular interval). Moreover, no cave zone indicators are reported in core reports except a possible cave fill slab sampled at 1492 ft (Figure 7.18). Since both core reports and well log analysis agree that there is no cave facies observed in this well, the modified criteria are verified.

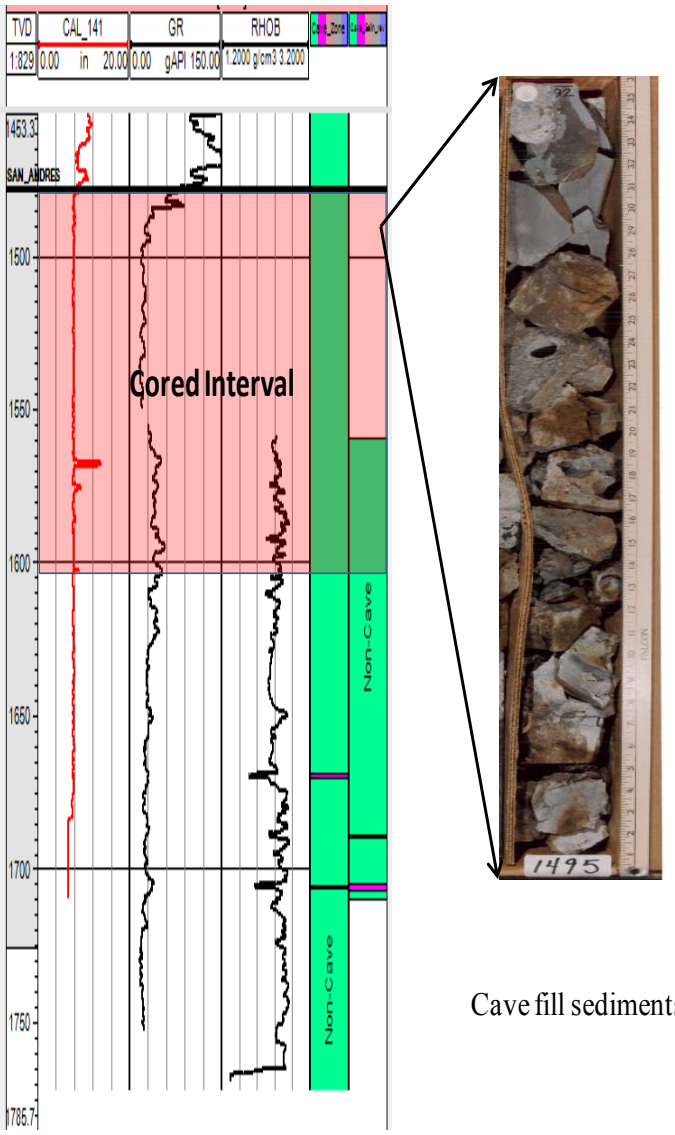


Figure 7.18- Core Sample from Well D. Core image is provided by KinderMorgan.

For Well D, although there is no core recovery in 1475-1490 ft interval, caliper and density log anomalies are observed at those depths. Above the "no core recovery zone" at 1472-1745 ft interval, core samples have high permeability (156-5570 md) and high porosity (0.17-0.30) and they are classified as packstone-grainstone. The predicted cave zone by the modified criteria is at 1486 ft within the no core recovery interval. Given the high permeability and porosity at depths adjacent to the specific interval of

interest, it is likely that the classification based on caliper and density log anomalies is accurate.

Although the remaining cored wells have no predicted cave intervals within the cored depths, they have caliper and density log anomalies above and below the cored interval and these anomalies are noted as cave indicator. Despite the anomalies, the modified criteria did not pick any cave zones at these wells due to high GR values. This situation might be due to the presence of cave fill sediments which are usually silt size materials with high GR. Since the modified criteria did not pick any cave zones at these wells and core descriptions also report no presence of cave facies, it is concluded that modified criteria is working properly.

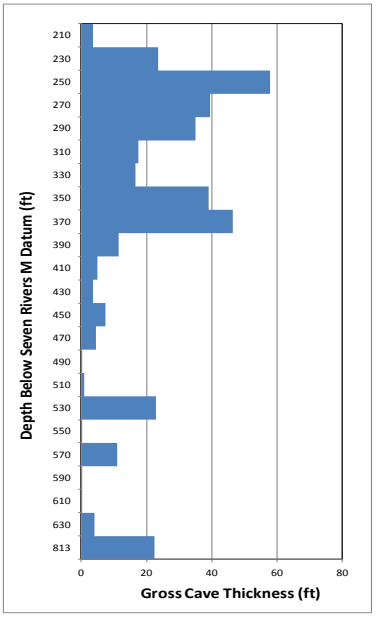
Several core descriptions and texture analyses for other wells are examined for further verification of the modified criteria. It is concluded that the proposed cave zone indicators are valid and provide accurate identification of cave facies. Therefore, the modified criteria are used for calculating cave facies thickness distribution within the selected region. The following section presents the analyses of thickness distribution of cave facies.

7.4 Cave Facies Thickness Distribution in Yates Field

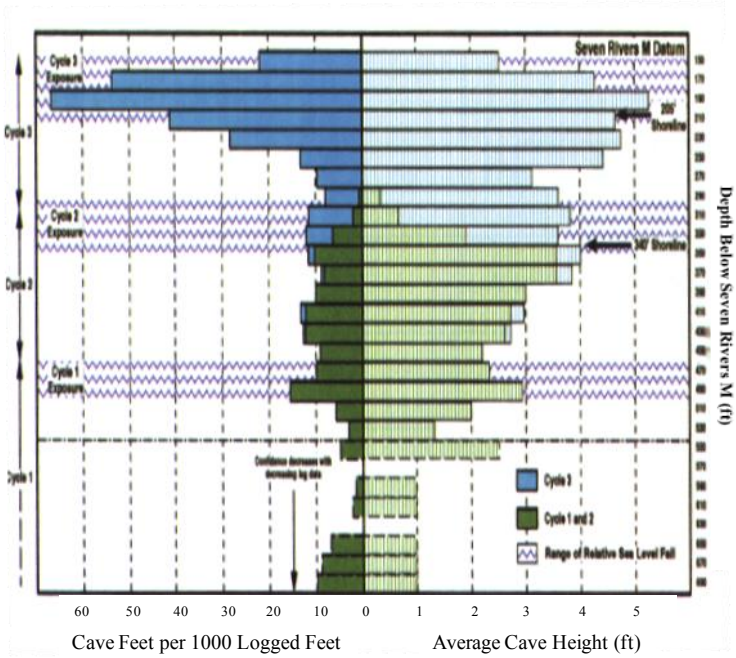
Cave network simulation of Yates field will be performed using conditioning data recorded at well locations. These conditioning data are in the form of an indicator variable representing the presence of a cave at that location. Moreover, to perform cave zone thickness simulation, thickness data are required at the conditioning locations. The cave zone thickness values inferred using the modified criteria are compared to the previously reported distribution by Tinker et al. (1995) for validation. The values calculated in this study are gross thickness estimates based on well logs. Modified criteria

(GR≤ 40 API, $\rho_{bulk} \leq 2.3$ g/cc and caliper deviation≥ 1.5 in) are used and gross cave thicknesses are determined.

Tinker et al. (1995) previously reported cave height variation as a function of depth below reference formation of Seven Rivers M datum (Figure 7.19b). For comparison, calculated gross cave thicknesses are also presented as function of the reference formation (Figure 7.19a).



a- Calculated Values



b- Distribution by Tinker et al. (1995)

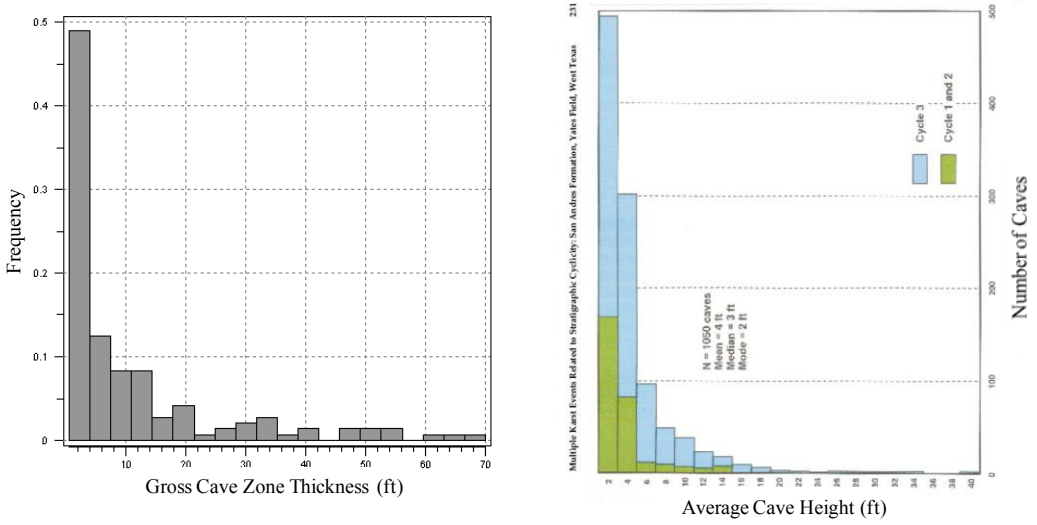
Figure 7.19- Comparison of Cave Height Distributions as a Function of Depth

It is observed that the calculated thickness values follow a similar trend as presented by Tinker et al. (1995). Further comparison is performed in terms of the cave height histograms (Figure 7.20a and b) and corresponding statistics (Table 7.2). It is observed that both distributions have similar trends. Tinker et al. (1995) studied the entire Yates field including the west side of the line of demarcation hence they reported more than 1000 cave intervals. Since only a small region is examined in this study (Figure 7.8),

identified cave zones are limited to 145 intervals. There is a difference in mean cave height values; the modified criteria has a larger mean compared to the value reported by Tinker et al. (1995). This difference is expected since thin cave intervals are lumped as a larger cave zone and the total thickness is identified as gross cave height in this study. Therefore, gross cave height values are larger than the net cave heights.

Table 7.2- Comparison of Cave Zone Thickness Statistics

	Modified Criteria	Tinker et al. (1995)
# of Cave Zones	145	1050
Mean (ft)	10.78	4
Median (ft)	4	3



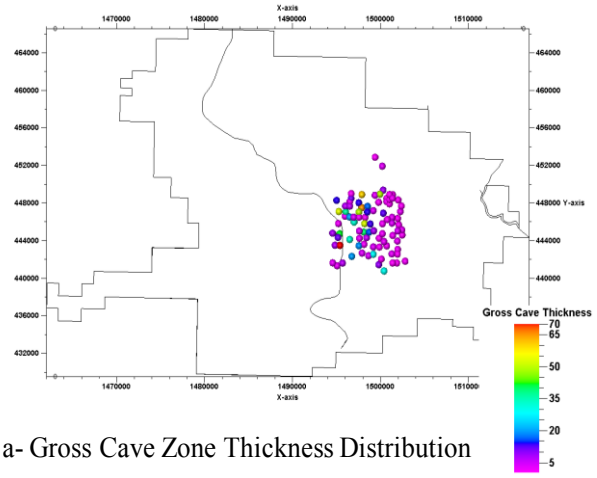
a- Gross Cave Zone Thickness Distribution b- Cave Height Distribution (Tinker et al., 1995)

Figure 7.20- Comparison of Cave Height Distributions.

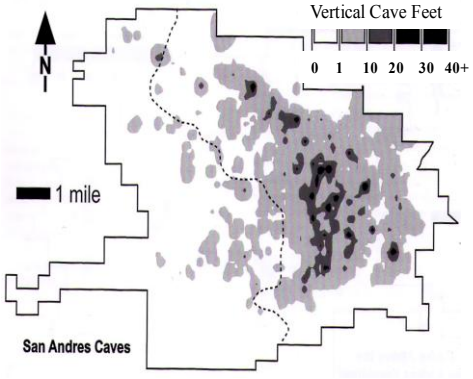
Stafford et al. (2008) presented spatial distribution of cavernous porosity in San Andres formation based on petrophysical analyses (Figure 7.21b). According Stafford et al. (2008), the cavernous porosity is identified by well logs with low bulk density values

or significant caliper divergences. They identified 1566 cave intervals with an average height of 3.9 ft and concluded that most of the cavernous porosity is clustered along the present crest of the San Andres formation.

Spatial distribution of cave zone intervals identified using the modified criteria are compared to the one proposed by Stafford et al. (2008) (Figure 7.21). It is observed that both distributions are in agreement despite the differences in cave heights. Because the cave zone height values reported in this study are given as gross thicknesses (i.e. thin cave intervals are lumped as a thicker cave zone), some of the cave height values do not agree especially in regions close to the line of demarcation (Figure 7.21) due to the abundance of small cave intervals at these locations.



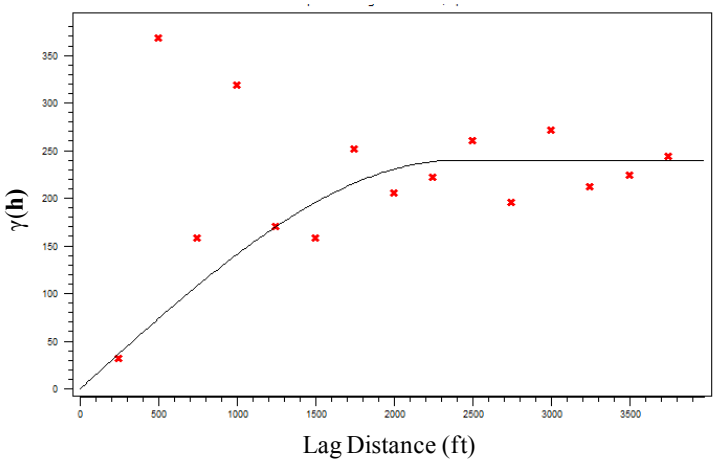
a- Gross Cave Zone Thickness Distribution



b- Distribution of Cavernous Porosity in San Andres Formation (Stafford et al., 2008)

Figure 7.21- Comparison of spatial distribution of cave zone in San Andres Formation

The variogram of the gross cave zone thickness is calculated and the corresponding model is implemented in the cave opening simulation algorithm. The experimental directional semi-variogram and the model fit are shown in Figure 7.22. The semi-variogram is modeled using a spherical model with sill contribution of 224 and a range of 2400 ft.



Azimuth=45° ; Dip=5°

Figure 7.22- Semi-variogram of gross cave zone thickness.

In conclusion, the cave zone thickness distribution is consistent with the previous studies (Tinker et al., 1995; Stafford et al., 2008) and the modified cave indicator criteria are valid and they yield valuable information on distribution of cave facies. These analyses result in identification of wells with cave zones. The inferred cave facies indicator and thickness data are used to condition the cave network simulation of Yates field. In the subsequent section, the cave network simulation results are presented.

7.5 Cave Network Simulation of Yates Field

The pattern simulation algorithm given in Chapter 4 is implemented to model the cave network system in the Yates field. The well locations with cave facies and identified cave zone thickness values are used for conditioning the pattern simulation that utilizes the MP-histograms constructed for Region 2 in Chapter 6. The motivation for using Region 2 as the training data set for the simulation of Yates field is discussed next.

Wind Cave and Yates field cave system are formed by similar mechanisms. According to Palmer and Palmer (2008), Wind Cave is an epigenic cave formed by mixing of two or more waters with different chemical compositions and major passages

follow the local dip direction. Tinker et al. (1995) proposed that the cave network in Yates field is also formed by mixing of waters with different chemistry. Moreover, Yates field cave system is recognized as a maze-network that is epigenic and formed by extensive dissolution of soluble rock along intersecting fissures. Although Wind Cave is located at the line end of a broad valley (Palmer and Palmer, 2008) and Yates field is formed in an island flank-margin (Tinker et al., 1995), the similarities in cave formation mechanisms justify the use of Wind Cave Region 2 point-set as a training image to model the cave network in Yates field.

To implement complex templates and corresponding MP-histograms of Region 2, spatial templates are resized and reoriented for Yates field. Resizing is based on well spacing and the spatial distribution of major fractures that have an influence on cave formation (Figure 7.23).

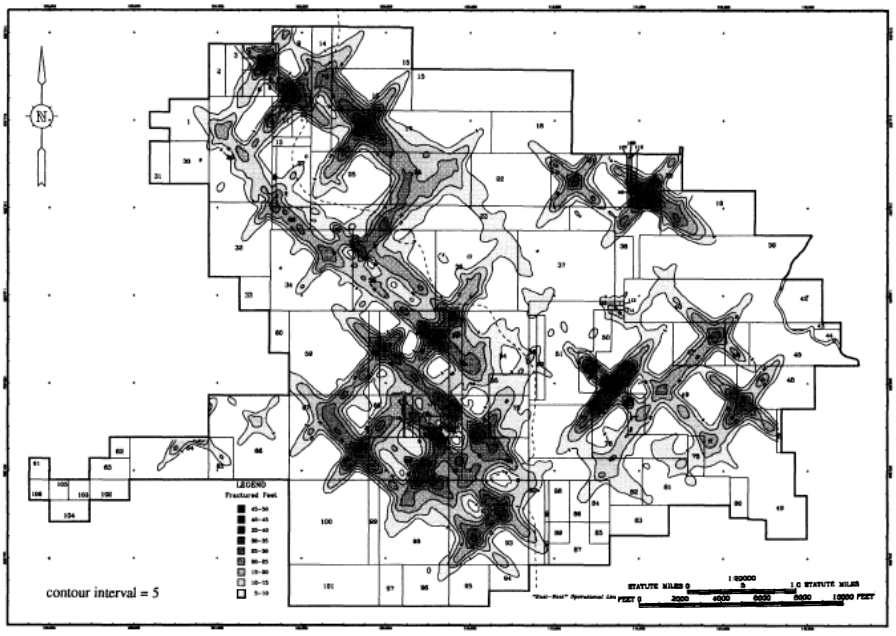


Figure 7.23- Spatial Distribution of Major Fractures. The contours are number of fractured feet in a 20 ft elevation slice map. Dark areas show high fractured feet (Tinker and Mruk, 1995).

Since the well spacing in Yates field is usually 10 acres, the average distance between wells is approximately 660 ft. Moreover, the extent of areas with high fracture density is determined to be around 600 ft (Figure 7.23). Therefore, complex spatial templates are resized with a lag distance of 600 ft. Since the major orientation of Wind Cave (NW-SE and NE-SW) is similar to the cave system trend in Yates field (Tinker and Mruk, 1995), spatial templates are slightly reoriented by varying the dip angle.

The pattern simulation algorithm is capable of locally varying template dip angle based on a given surface dip map. This will enable the modeling of network orientation accurately. For this purpose, the dip map of San Andres surface top is implemented in network simulation. The dip map is constructed in GOCAD by smoothing the top surface of the San Andres formation and calculating surface dip angle (Figure 7.24). Then, the map is implemented in cave network simulation for re-orientation of the templates based on the spatially varying dip angle map.

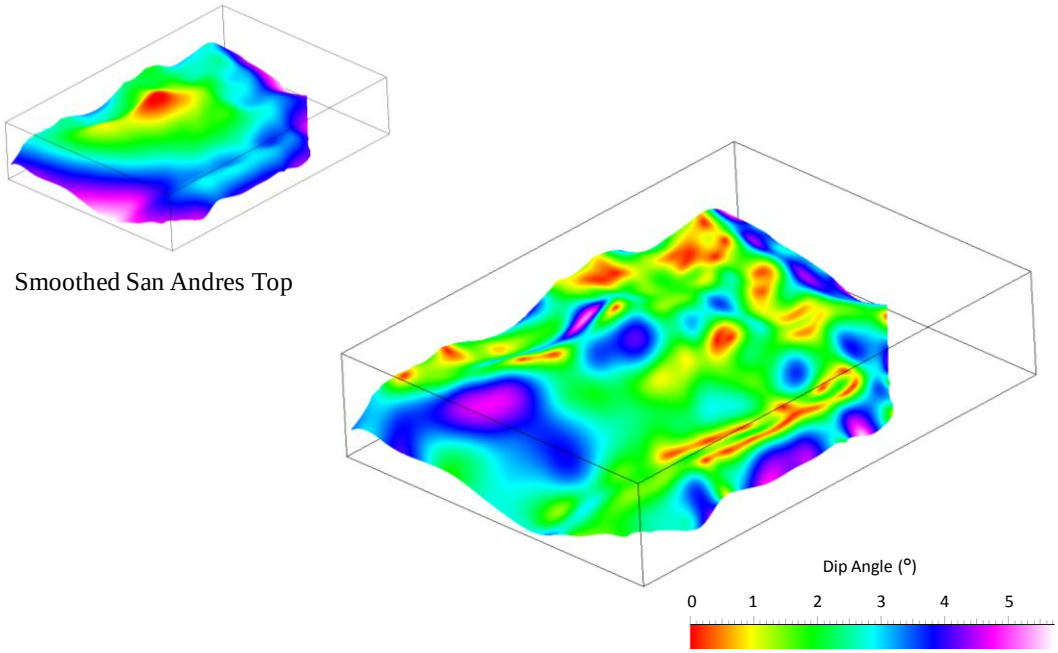
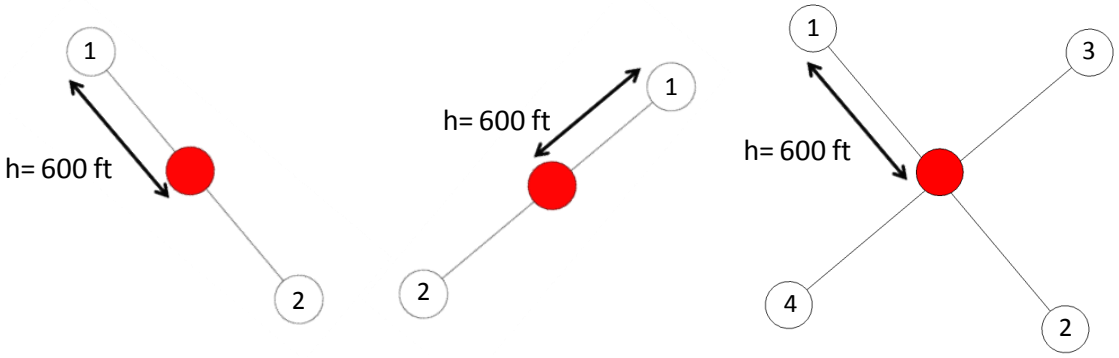


Figure 7.24- Dip Angle Map of Smoothed San Andres Formation Top (Vertical Exaggeration x10)

7.5.1 Pattern Simulation with Multi-Node Templates

For pattern simulation of cave network with multi-node templates, 3 different spatial templates are selected; two of them are for sequential pattern simulation and the third one is a complex 4-node template (Figure 7.25). These templates are previously used in network simulation Wind Cave. Template properties and pattern histograms are given in Table 7.3 and Figure 7.26. It is observed that Templates 1 and 2 mostly report simple linear patterns whereas Template 3 captures relatively complex non-linear patterns. Template 3 pattern histogram is further refined by eliminating some of the simple linear configurations that have already been captured using Templates 1 and 2.

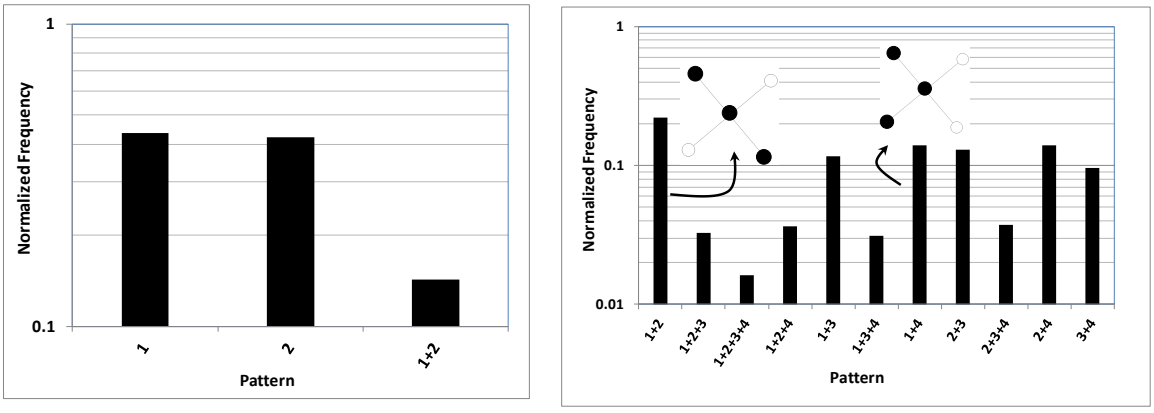


a- Template 1 b- Template 2 c- Template 3

Figure 7.25- Templates Used in Pattern Simulation of Yates Field

Table 7.3- Properties of Spatial Templates for Yates Field

Template	Azimuth Angle (°)	Dip Angle (°)
1	-40	5
2	50	5
3	-40, 50	5

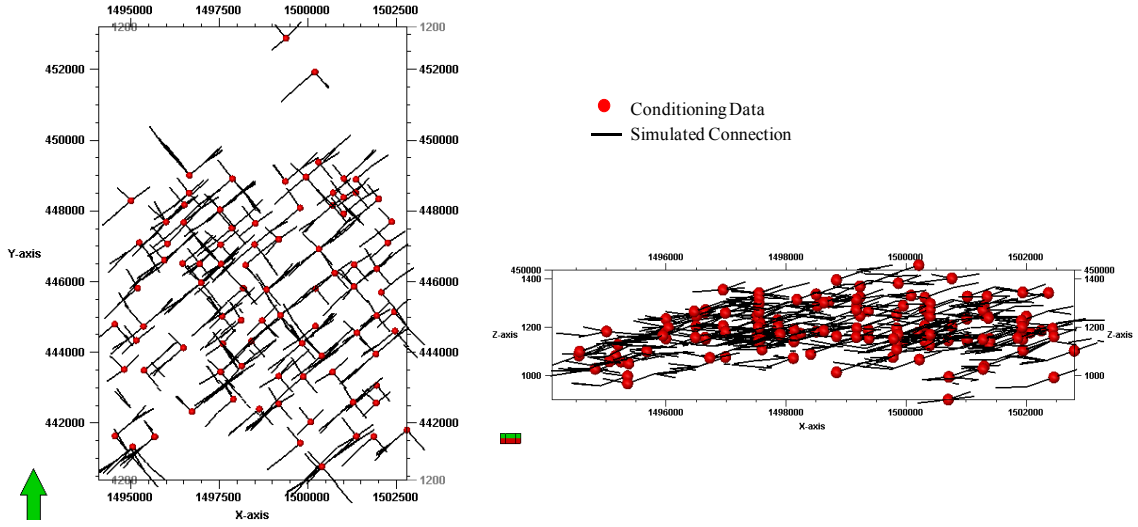


a- Templates 1 and 2

b- Template 3

Figure 7.26- Pattern Histograms for Yates Field. These histograms are obtained by scanning the point set data for Region 2 of Wind Cave using templates shown in Figure 7.25.

Initially, two different realizations are obtained for the cave network system in Yates field. Realization 1 is simulated by sequentially using Templates 1 and 2; first patterns are simulated with Template 1 and in the second step Template 2 is implemented. Tolerance window is defined by azimuth angle tolerance of 20°, dip tolerance of 3° and a lag tolerance of 1/3. In both steps, network growth is terminated at 350 iteration steps. The simulated cave network for Realization 1 is illustrated in Figure 7.27. It is observed that the simulated network is also a maze-type network of caves and it honors the major trend in predicted Yates field by Tinker and Mruk (1995).



Top View

Cross Section (Vertical Exaggeration x4)

Figure 7.27- Realization 1 for Yates Field. (Arrow shows the North direction.)

Realization 2 shown in Figure 7.28 is obtained by simulating the network using Template 3 with same the tolerance window as Realization 1. The pattern growth is terminated at 500 simulation steps to ensure that all nodes around the conditioning locations are implemented as "simulation node". Since increasing the number of simulation steps will increase the number of simulated nodes while resulting in an exhaustive set of cave passages, the number of simulation steps selected is limited to 500.

The simulated cave network also has a maze-type structure with intersecting features. Realization 2 has longer continuous passages compared to Realization 1 and this is expected since Realization 1 is obtained by sequential simulation of patterns using two different templates and that can result in termination of some passages at the end of the first step. On the other hand, in Realization 2, patterns continue to grow until the maximum simulation step is reached. Moreover, intersecting cave passages are more prevalent in Realization 2 mainly due to the structure of Template 3 that has 4-nodes in two intersecting directions. Although Templates 1 and 2 used for the previous realization are in intersecting directions, when applied sequentially patterns might not intersect at some locations unless those locations are selected as simulation node.

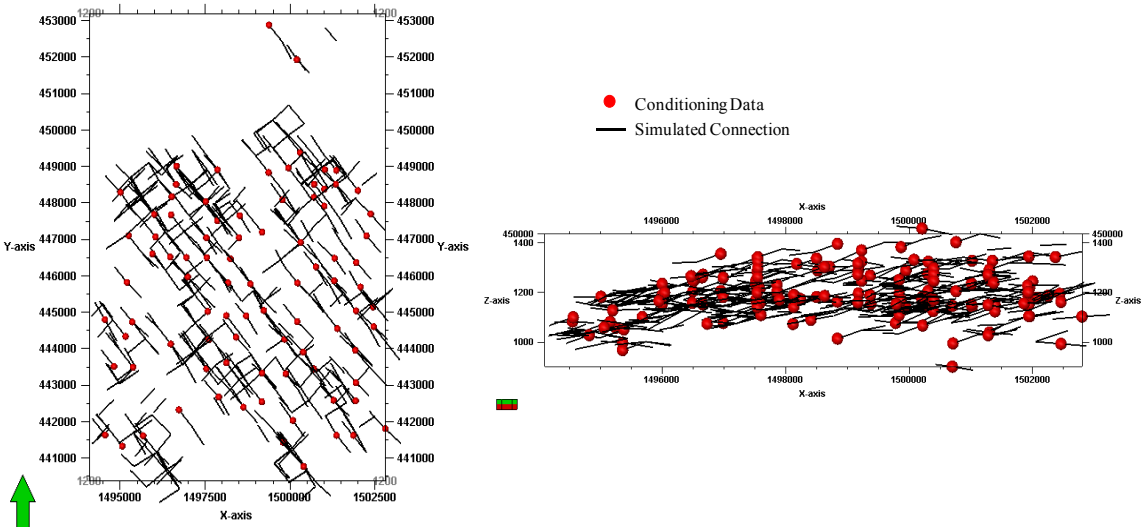


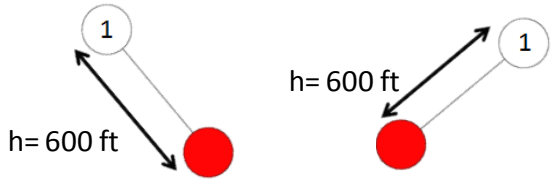
Figure 7.28- Realization 2 for Yates Field obtained using the multi-directional, multimode template in Figure 7.25c.

The cave network simulation results obtained using MPS are later compared against a simulation constrained to only two-point statistics algorithms in the form of

indicator semi-variograms. This indicator variogram-based approach would represent the standard approach to modeling karst systems commonly practiced in industry.

7.5.2 Pattern Simulation with Single Node Templates

The third realization is obtained by sequentially implementing two single node templates (Figure 7.29) and corresponding pattern histograms which are constructed by using the training image with an azimuth tolerance of 10°, dip tolerance of 10° and lag tolerance factor of 1/3. Since the template has single node, pattern histogram only reports the number of captured connections (Table 7.4). The simulation is performed sequentially implementing templates Single 1 and 2 with the same tolerance window used in Realizations 1 and 2. In both simulation steps, network growth is terminated at 500 steps and it is observed that pattern growth is limited due to very simple structure of the template.



a- Single-1 b- Single-2

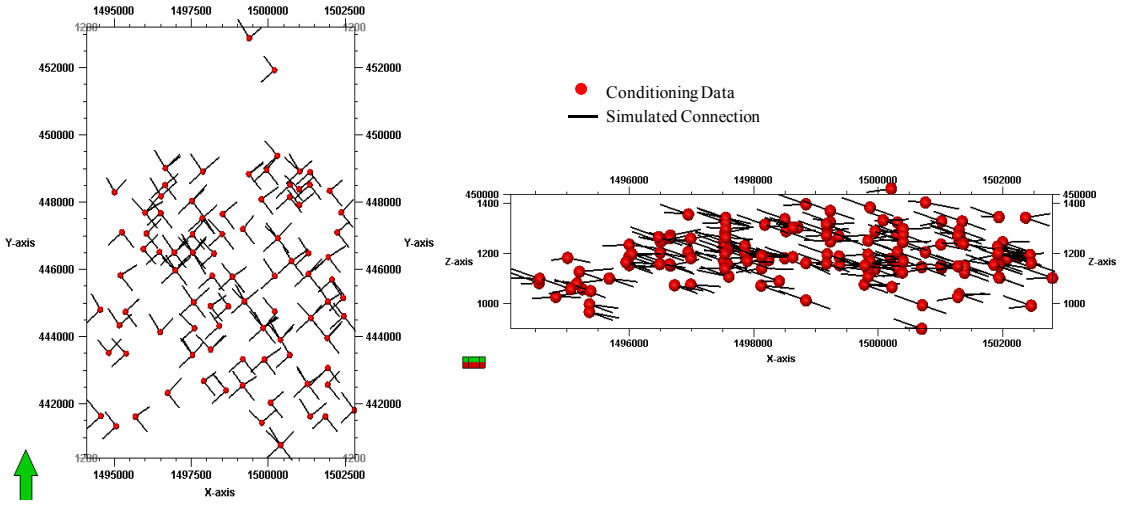
Figure 7.29- Single Node Templates

Table 7.4- Properties of Single Node Templates

	Azimuth Angle (°)	Dip Angle (°)	# of Occurrences
Single 1	-40	5	1144
Single 2	50	5	891

Realization 3 is illustrated in Figure 7.30 and it is observed that use of single node templates results in short and generally disconnected passages. This is mainly due to the

template configuration with a single node. Moreover, the simulated network has many intersecting passages that are due to the sequential implementation and simulating the passages using single node templates.



Top View

Cross Section (Vertical Exaggeration x4)

Figure 7.30- Realization 3 with Single Node Templates

7.5.3 Comparison of Simulated Cave Zone Thickness Distributions

Along with the simulation of the pattern of caves in Yates field, spatial distribution of cave zone thickness is also simulated. For this, the cave thickness data along wells are used for conditioning and the semi-variogram model given in Figure 7.22 is utilized to describe the spatial continuity of the cave zone thickness and Figure 7.31 illustrates the simulated cave zone thicknesses in Realization 1. It is observed that the thickness is larger in locations with relatively thicker cave zone whereas thinner cave intervals are clustered in regions with lower cave zone thickness as expected.

The simulated cave height distributions are compared to the original thickness distribution obtained in Section 7.4 and corresponding statistics are given in Table 7.5 and Figure 7.32.

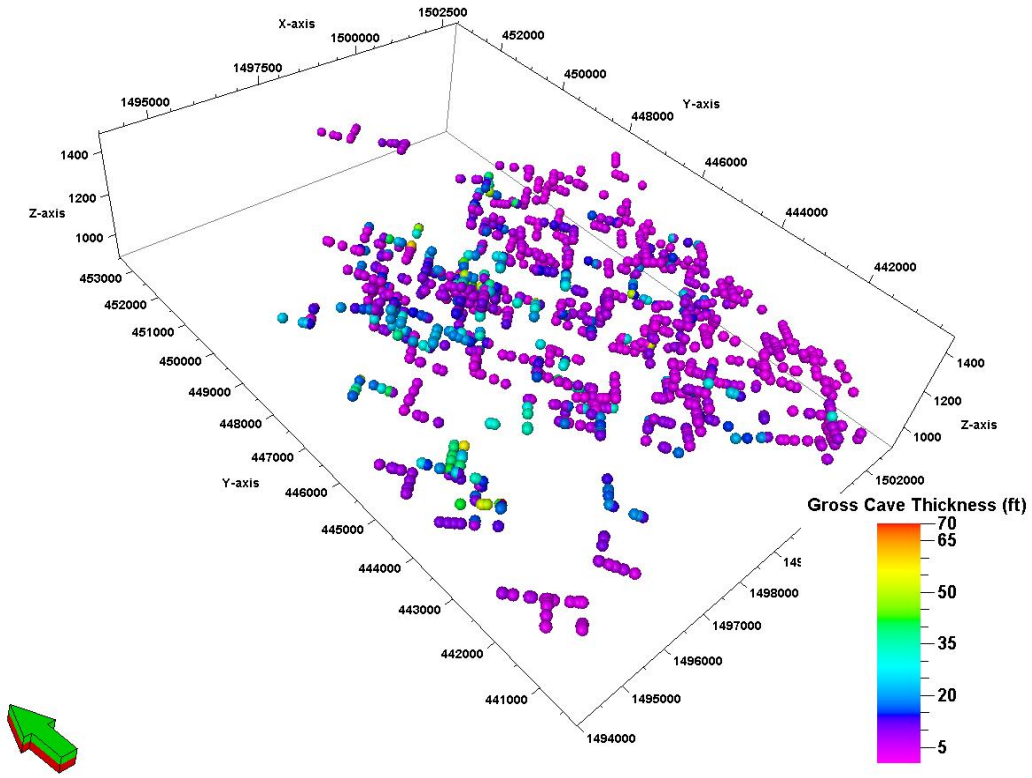


Figure 7.31- Simulated Cave Zone Thickness Distribution in Realization 1. Warmer colors indicate thicker cave zone. Arrow shows the North direction.

Table 7.5- Comparison of Simulated Cave Zone Thickness Statistics for the Yates Field

	# of Data	Mean (ft)	Median (ft)	Standard Deviation (ft)
Conditioning Data	145	10.78	4	15.00
Realization 1	3589	9.42	5.38	11.00
Realization 2	2543	9.45	5.78	9.97

The tabulated results indicate that the simulated cave networks reproduce the statistics of the conditioning data. The statistics shows that mean and median values are close to each other and simulation results are consistent. The deviation from the target statistics is owing to ergodic fluctuations.

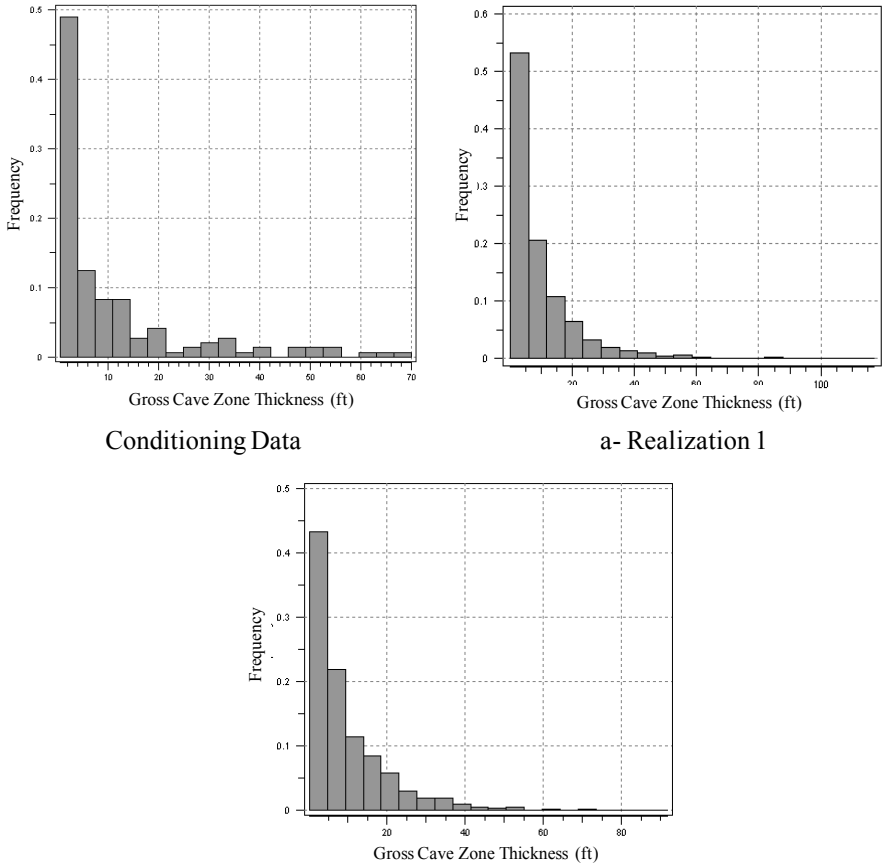


Figure 7.32- Simulated Cave Zone Thickness Distributions for the Yates Field

After the modeling of cave network system in Yates field, Realization 2 is implemented in a flow simulation model. In the following sections, the flow-modeling attempt is discussed.

7.6 INTEGRATION OF CAVE NETWORKS IN FLOW SIMULATION MODELS

Simulated cave networks in Yates field are integrated into a flow simulation model developed using a commercial black-oil simulator, CMG-IMEX. The objective of flow modeling is to demonstrate the importance of modeling spatial continuity of the karst network accurately using the MPS algorithm developed in this research. Comparison will be made against the flow responses for a reservoir model developed

using an indicator simulation approach which would be the common approach for karst modeling adopted currently.

First, a 3D grid system is constructed between the San Andres formation top shown in Figure 7.33 and the well bottom depth surface. The model consists of 103*135*10 grid blocks.

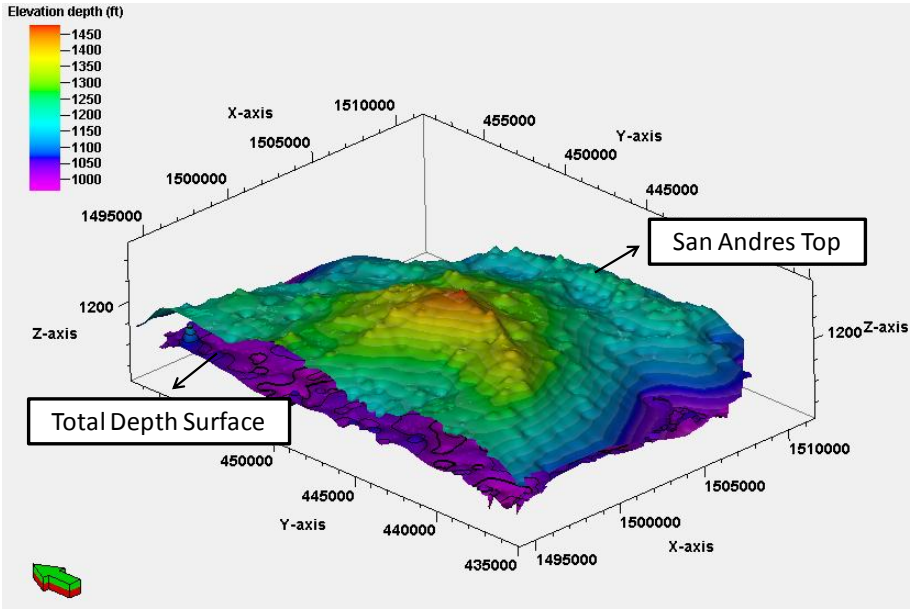


Figure 7.33- Formation Top Surfaces. Green arrow shows the North direction.

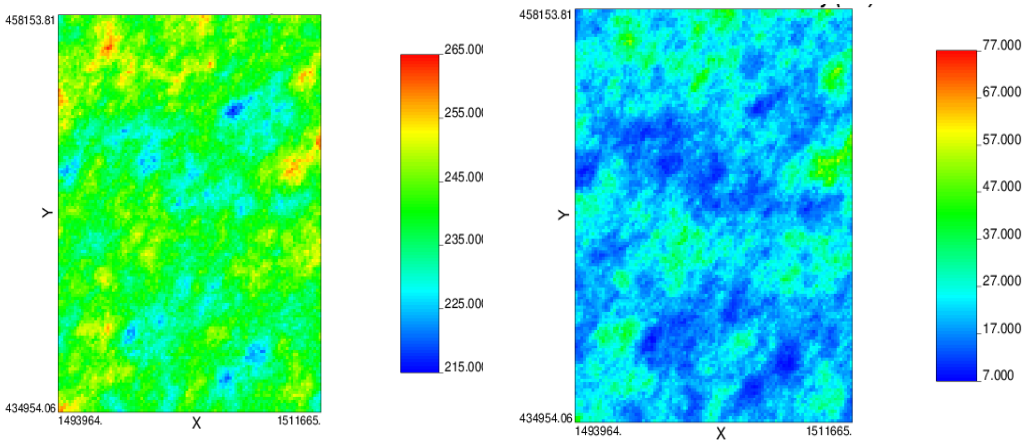
Cave zone thicknesses are represented by varying the grid thickness in the z-direction for the grids with cave facies. Parameters used in flow simulation are obtained from the literature (Gilman et al., 1995; Cheng and Kwan, 2012). The parameters used for flow simulation are summarized in Table 7.6. Petrophysical properties, reservoir fluid characteristics (oil viscosity, oil density, bubble point pressure and temperature) and operating conditions are previously reported by Gilman et al. (1995) and Cheng and Kwan (2012). Initial reservoir conditions (i.e. depth to water-oil-contact, initial pressure

at a reference depth) are obtained from the internal reports provided by KinderMorgan Corporation.

Table 7.6- Parameters Used in Flow Simulation

Number grids in x-direction	103
Number grids in y-direction	135
Number grids in z-direction	10
Grid block dimension-x (ft)	171.53
Grid block dimension-y (ft)	171.53
Grid block dimension-z (ft)	Variable
Porosity	0.17
Rock compressibility (1/psi)	1E-6
μ_{oil} (cp)	6.5
ρ_{oil} (g/cc)	0.85
B_o (rb/STB)	1.056
$P_{bubble\ point}$ (psi)	450
$T_{reservoir}$ (°F)	82.4
$P_{initial}$ (psi) @ 1445' depth	800
Water-Oil-Contact (ft)	1695

In the flow simulation model, permeability is used as a history match variable for honoring the primary production data. For cave and non-cave facies, permeability models obtained by *Sequential Gaussian Simulation* program of SGEMS (Remy et al., 2008) are used (Figures 7.34a-b). These realizations are conditioned to the core data and gross cave zone thickness semi-variogram given in Figure 7.22 is used in the simulation.



a- Cave Facies Permeability Field

b- Non-Cave Facies Permeability Field

Figure 7.34- Permeability Fields used in Simulation Models

Relative permeability curves are constructed in CMG assuming a mixed-wettability rock (Figure 7.35) and the end point relative permeabilities given in Gilman et al. (1995) are implemented.

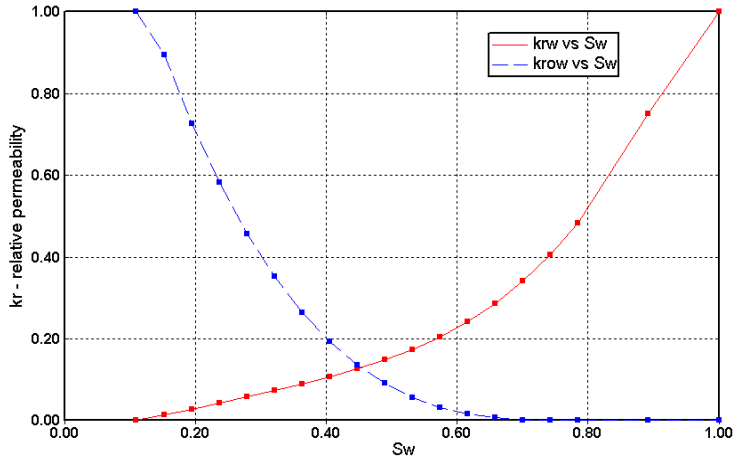


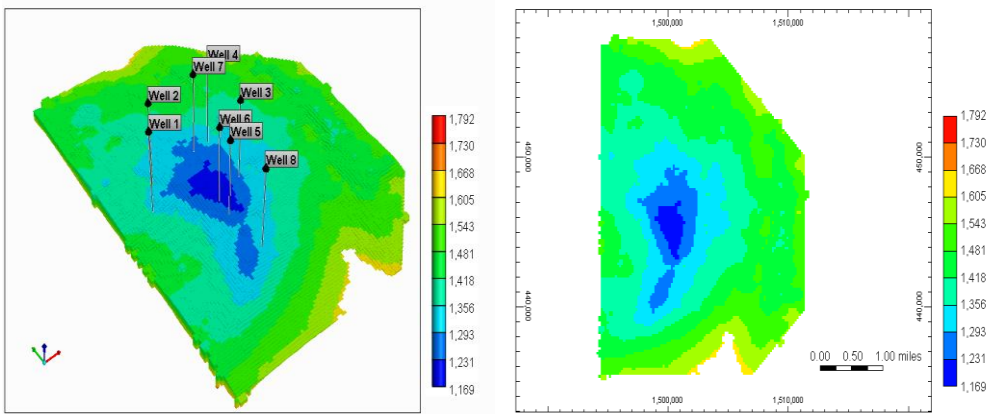
Figure 7.35- Relative Permeability Curves used for flow simulation.

The flow simulation is performed using 8 wells with field production history and the primary drainage period of 1927-1974 is considered only. Since most of the wells

during the early development of the field were open-hole, the casing size information of each well is assigned as the well bore radius and the perforation depths reported in well logs are specified.

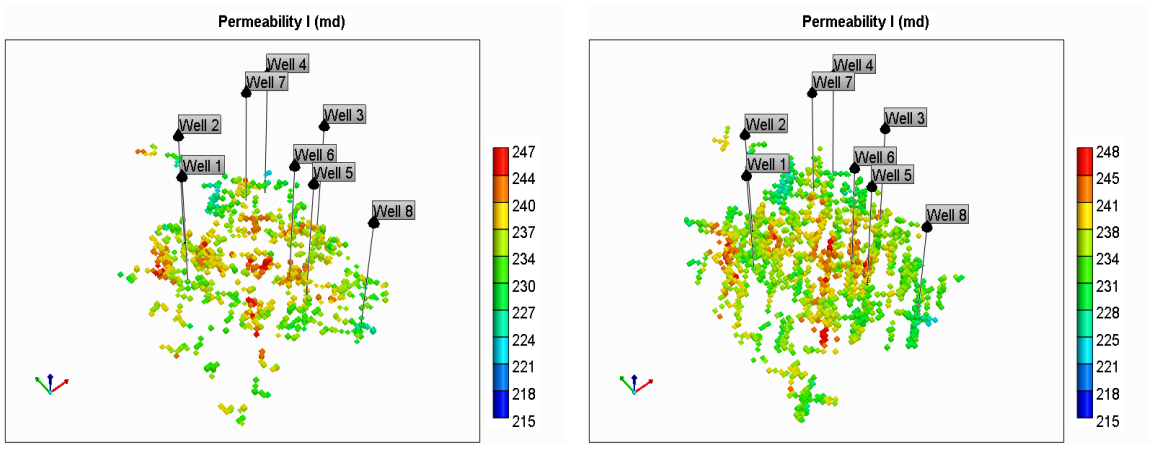
7.6.1 Analysis of Flow Simulation Results

Prior to analyzing the flow simulation results, calibration of the flow model against the available primary production data was attempted. As mentioned previously, the cave networks corresponding to MPS Realizations 1 and 2 were used as the geological models. The models as they are represented in the flow simulator are shown in Figures 7.36 and 7.37. In order to visualize the cave facies better, grid permeabilities less than 215 md are not shown in Figure 7.37.



a- 3D View b- Top View

Figure 7.36- Flow Simulation Model. Colormap shows depth-to-grid top values in feet. The model is similar for both Realizations 1 and 2, except they have different grid thickness and grid permeability distributions based on the corresponding cave network.



a- Realization 1

b- Realization 2

Figure 7.37- Spatial Distribution of Cave Facies in Flow Models 1 and 2. Cave facies are represented by high-permeability grids. Permeability values are before history matching. Realization 1 is obtained by sequential pattern simulation whereas Realization 2 is constructed by only using 4-node spatial template.

By using these cases, calibration of the flow model against the available primary production data was attempted and the results are compared (Figures 7.38-7.39). It is observed that for some of the wells, simulated cumulative production values are higher than the field data. Although the simulation results are close to the primary production in Wells 3 and 6, the flow models should be calibrated by altering the permeability locally.

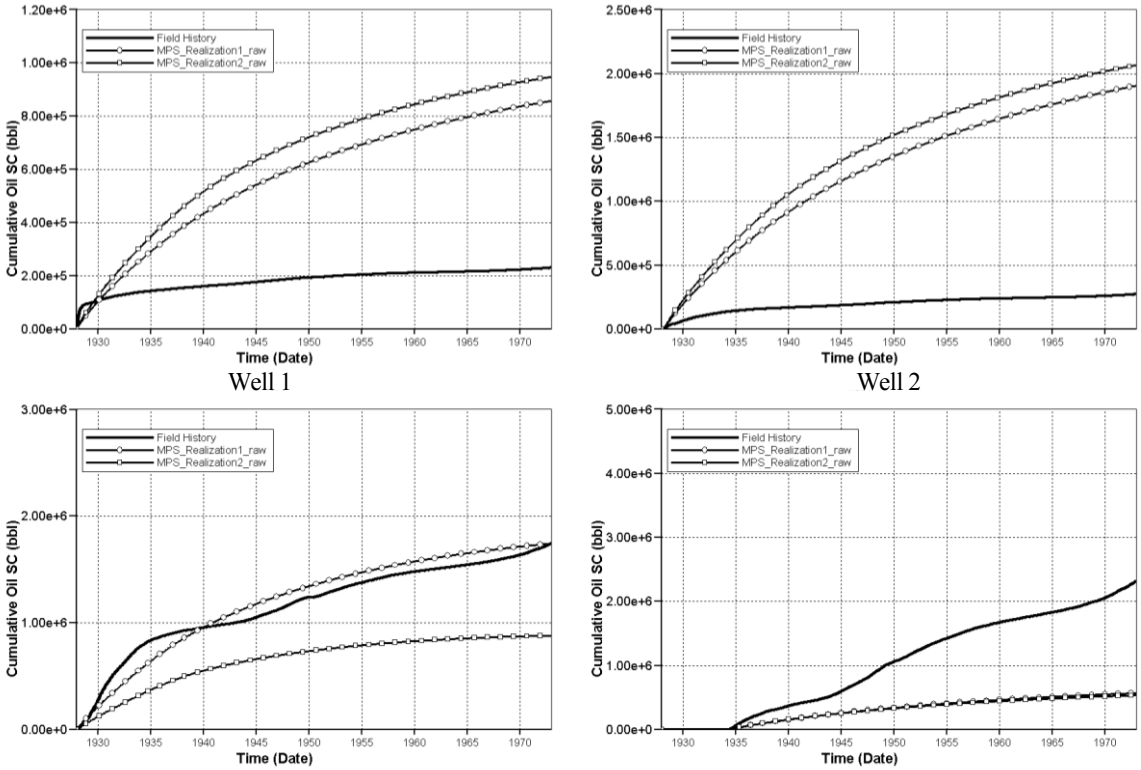


Figure 7.38- Comparison of Flow Simulation Results before Calibration for Wells 1-4

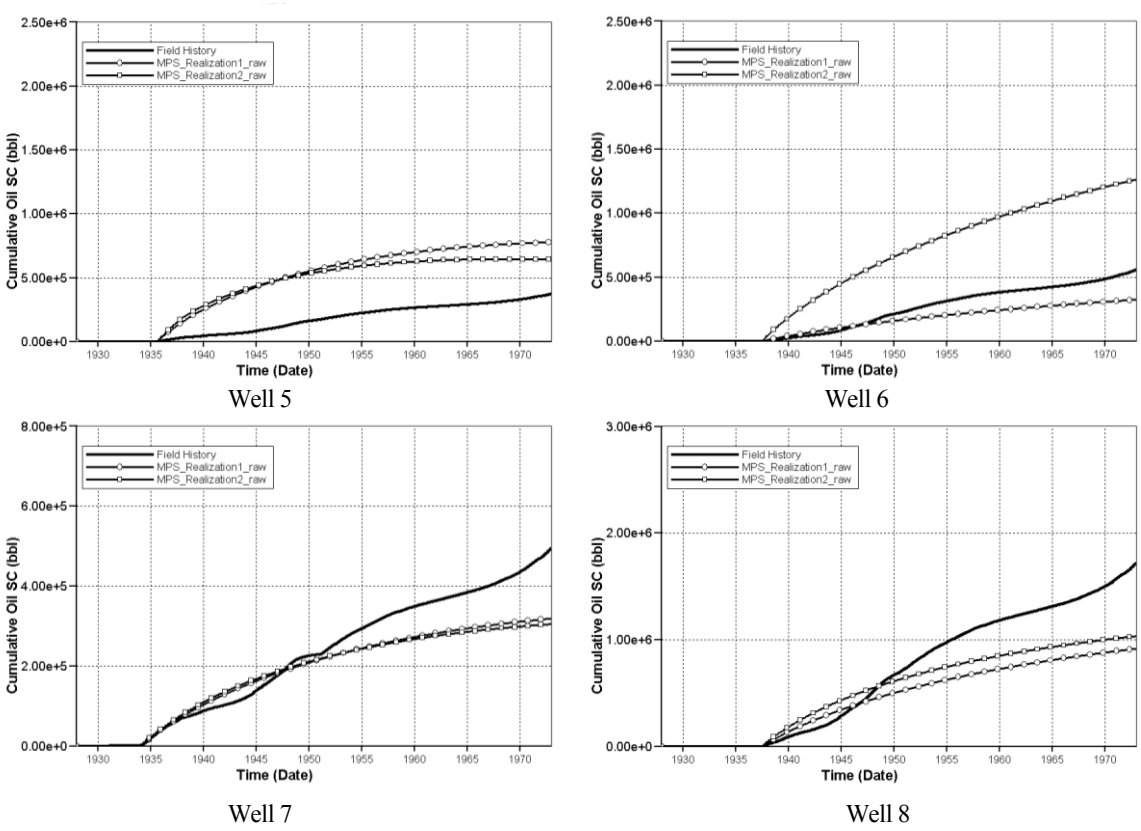


Figure 7.39- Comparison of Flow Simulation Results before Calibration for Wells 5-8

Permeability values of both cave and non-cave grids are altered around the wellbores for history matching purposes. Figures 7.40 and 41 show that permeability might change drastically along the well bore in the San Andres formation. This permeability variation might be due to dissolution features and the chaotic events associated with cave collapse. Therefore, grid permeability values around the wellbore are changed to match the oil production history for each well. Matrix permeability is varied from 5 md to 50 md, and cave facies permeability is perturbed in the range of 75 md- 250 md. These ranges are based on the core data given in Figures 7.40-7.41.

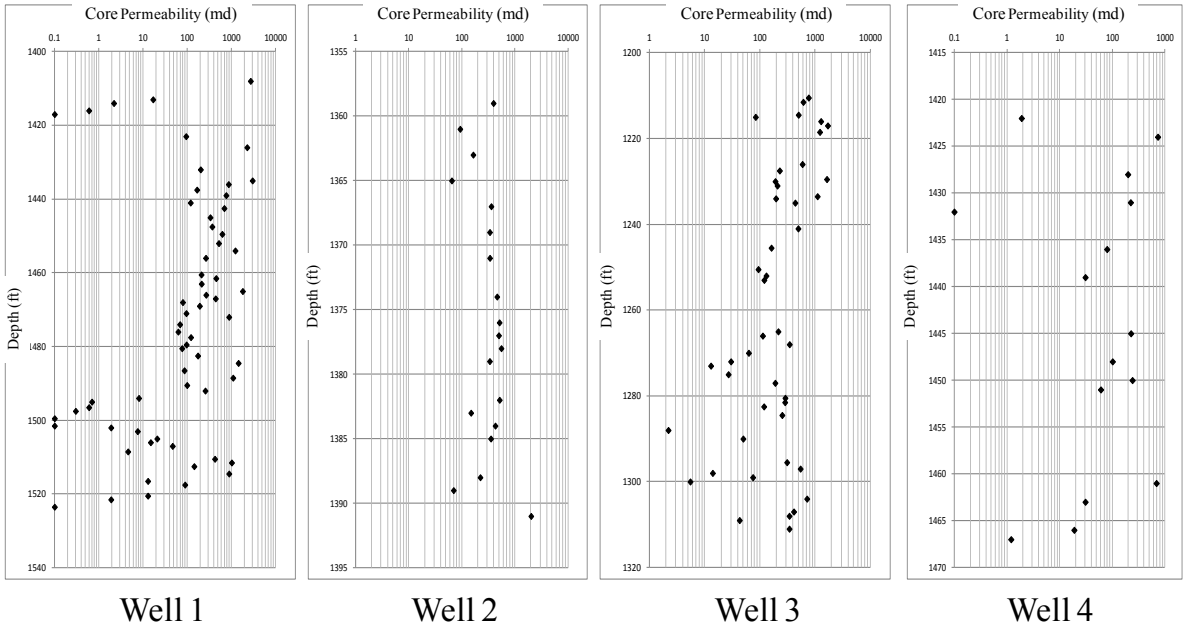


Figure 7.40- Core Permeability Variation as a function of Depth for Wells 1-4

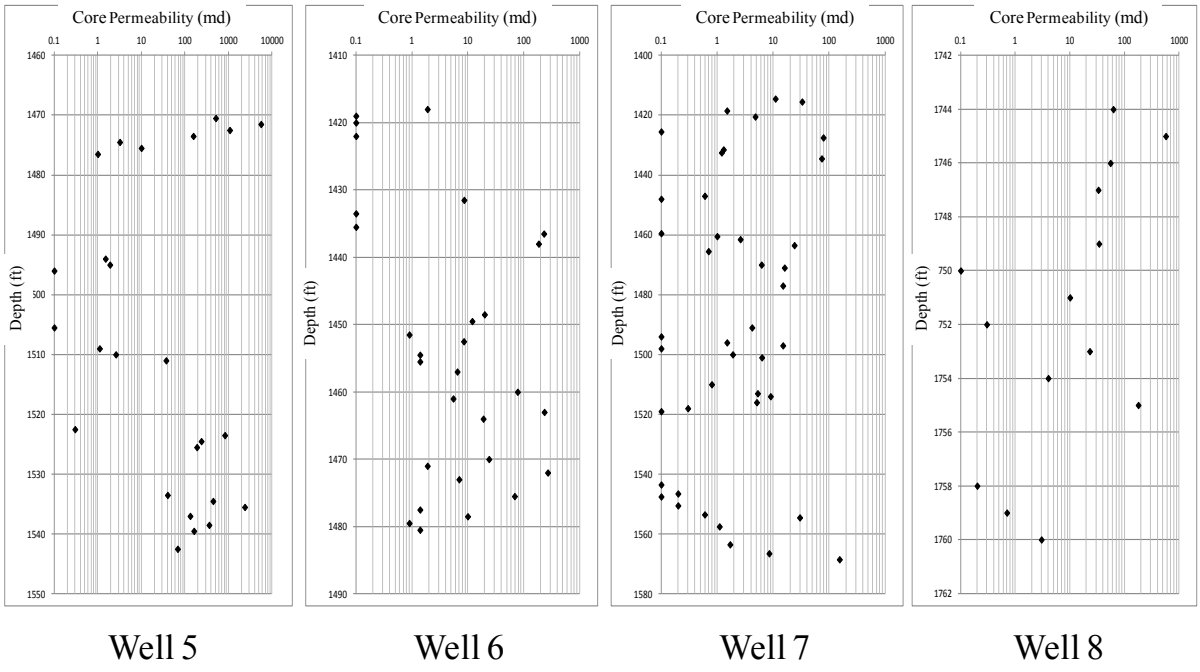


Figure 7.41- Core Permeability Variation as a function of Depth for Wells 5-8

Results obtained after the model calibration are shown in Figures 7.42-7.43. For better comparison, the simulation results before the calibration are also given in these figures. It is observed that oil production profiles are matched successfully for most of the wells. Although in some wells (Wells 1, 7 8) early production data has some mismatch, the overall oil production profiles are in agreement.

Since the main objective is to demonstrate the importance of correctly representing the connectivity of the karst network, only the oil production profiles are considered.

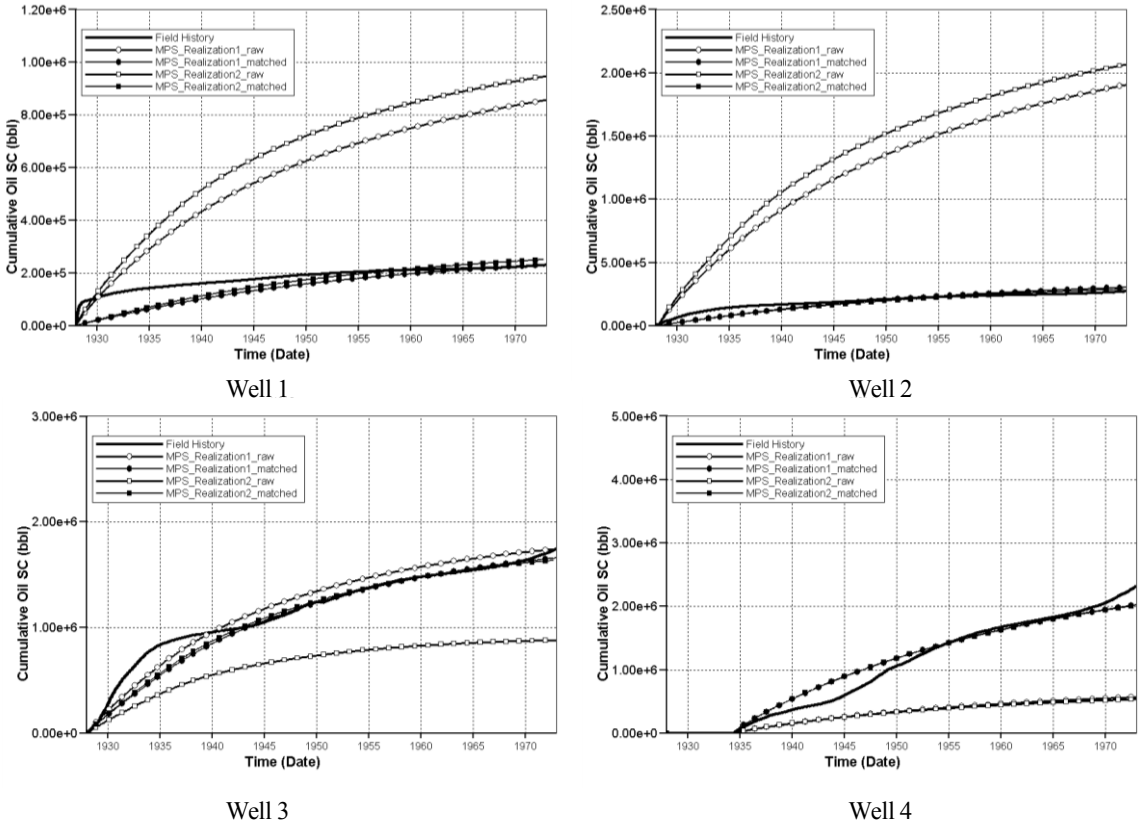


Figure 7.42- Flow simulation results obtained after model calibration for Wells 1-4. For comparison, the base case simulation results are also shown (Solid Line: Field History).

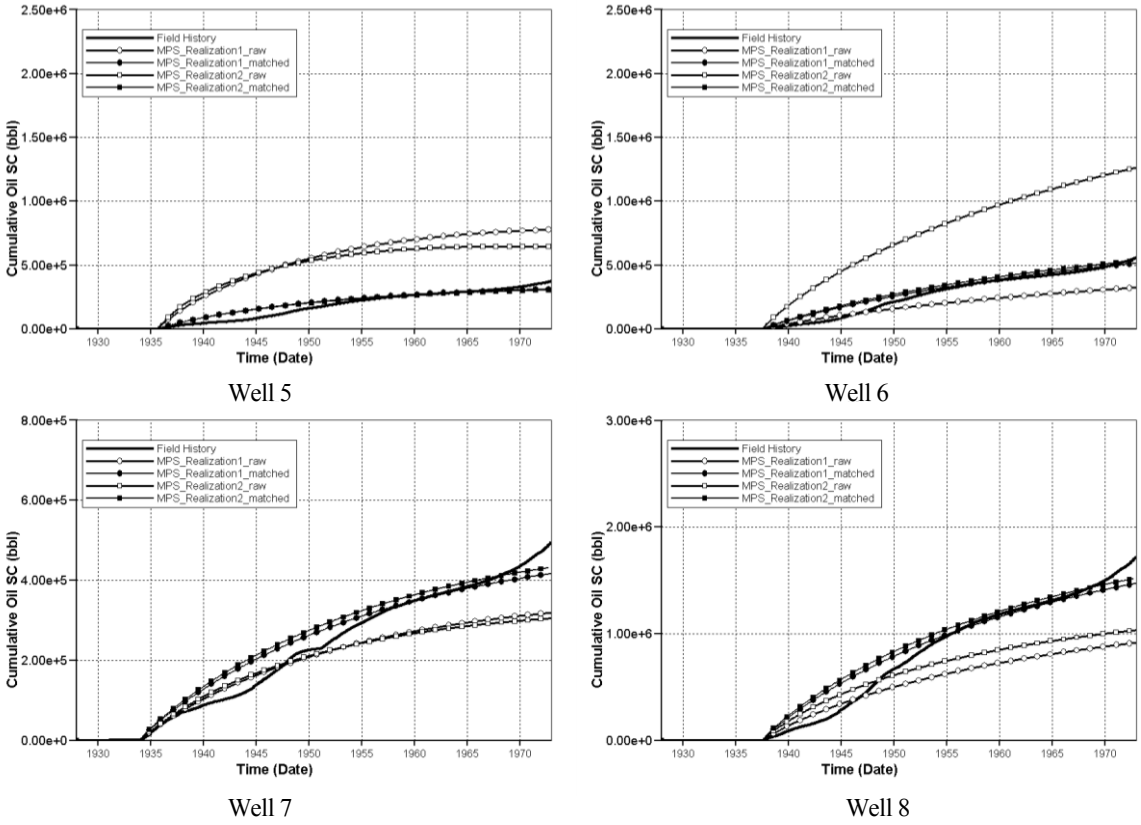
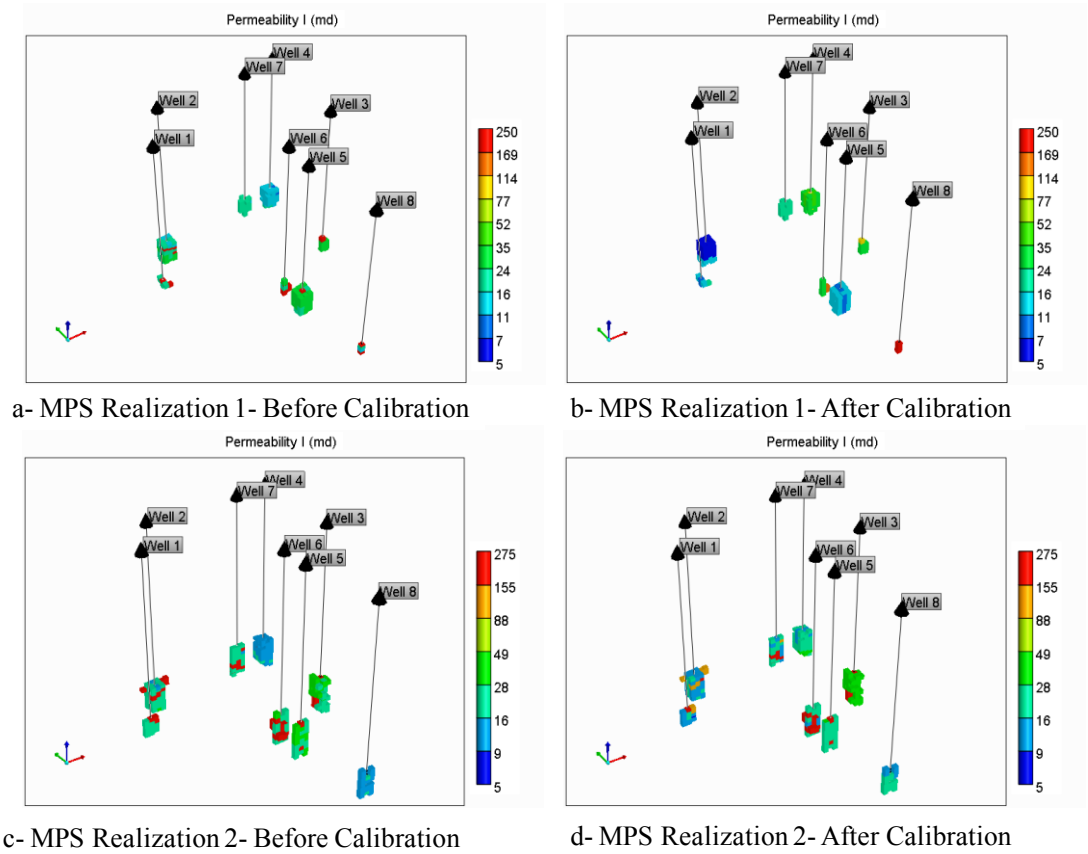


Figure 7.43- Flow simulation results obtained after model calibration for Wells 5-8. For comparison, the base case simulation results are also shown (Solid Line: Field History)

Models are calibrated by only modifying the grid permeability values around the wellbores. Permeability alterations are mainly based on the core data given in Figures 7.40-7.41. In Figure 7.44, grids with permeability modifications are shown. Permeability values are compared before and after the model calibration for both MPS Realization models 1 and 2. Permeability is reduced in some locations. This situation might be due to the presence of low permeability facies (i.e. cave fill sediment). Since cave network simulation results in location of cave central line and corresponding extent of cave facies, it is liable to have cave fill sediments, which have low permeability, within that particular cave zone.



a- MPS Realization 1- Before Calibration

b- MPS Realization 1- After Calibration

c- MPS Realization 2- Before Calibration

d- MPS Realization 2- After Calibration

Figure 7.44- Variations of Permeability in the Calibrated Models. Grids with permeability modification match are shown only.

The calibration using primary production data demonstrates that the pattern simulation algorithm yields detailed description of spatial distribution of cave facies in flow models. Although there is no information on well operating condition or any underlying porosity map, history match is successfully obtained by only varying permeability values around the wellbore. After this basic calibration of model using the primary production data, we are now ready to explore the impact of accurately representing network connectivity on flow responses. For this, a water injection case will be simulated as discussed next.

7.6.2 Importance of Cave Network Description for Predicting Fluid Flow Responses

In order to demonstrate the importance of describing cave network connectivity on fluid flow, several simulation models are constructed for a water injection case. The flow simulation models obtained by Realizations 1 and 2 (Figure 7.36) are implemented without the calibrated permeability values. Also, for this demonstration, a small region of interest is selected around the production wells (Figure 7.45).

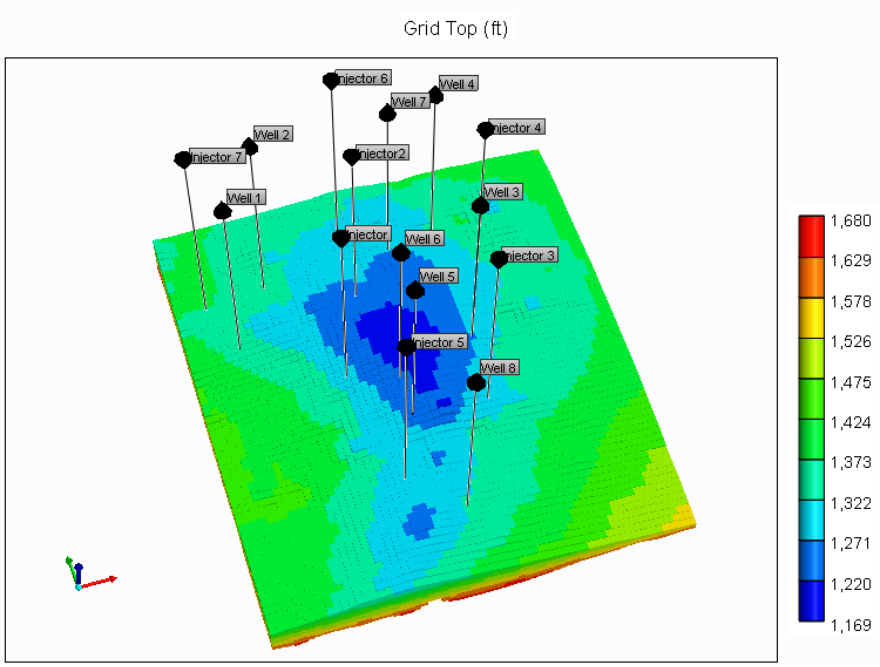


Figure 7.45- Region of Interest for Water Injection Demonstration

For comparison of fluid flow responses obtained using a more traditional modeling workflow, three different flow models are constructed using the cave facies distributions obtained by applying *Sequential Indicator Simulation (SISIM)* program of SGEMS (Remy et al., 2008). SISIM realizations are conditioned to the facies logs constructed by using the modified criteria described in Section 7.3. First, semi-variograms of the cave facies indicator data is calculated (Figure 7.46) and it is modeled using a spherical model with a maximum range of 8910 ft and a sill contribution of 0.06

along 105° azimuth and 5° dip directions. The range of the semi-variogram in the minimum continuity direction is 1440 ft.

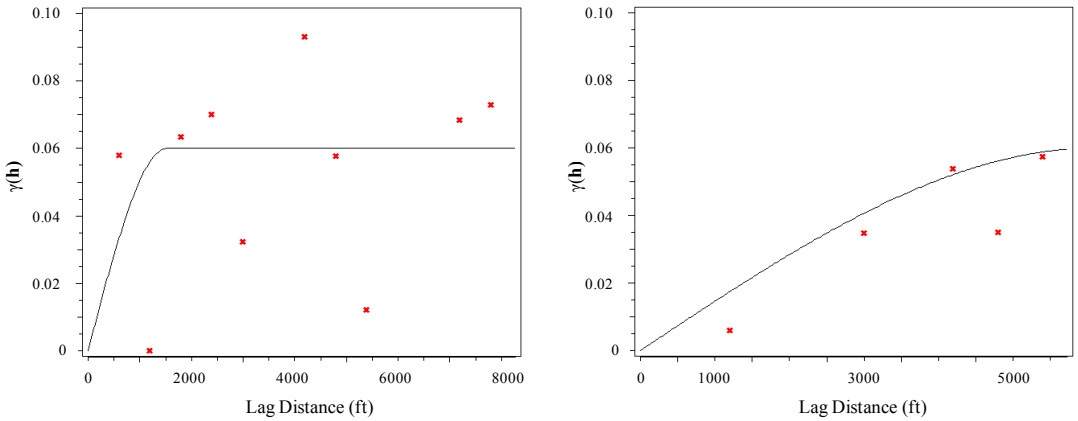


Figure 7.46- Directional Semi-variograms of Indicator Data for Cave Facies.

Once the semi-variogram model is obtained for the cave facies indicator, it is implemented into *SISIM* program of SGEMS (Remy et al., 2008) to obtain several realizations for spatial distribution of cave facies. The realizations are obtained by using a search ellipsoid with a maximum radius of 20,000 ft along 105° azimuth and 5° dip angles. A maximum number of 200 conditioning data is assigned for the kriging system and three realizations are shown in Figure 7.47. Although the simulated cave facies follow the major connectivity direction, they are more diffused compared to the simulated cave facies obtained using non-gridded MPS given in Figure 7.37.

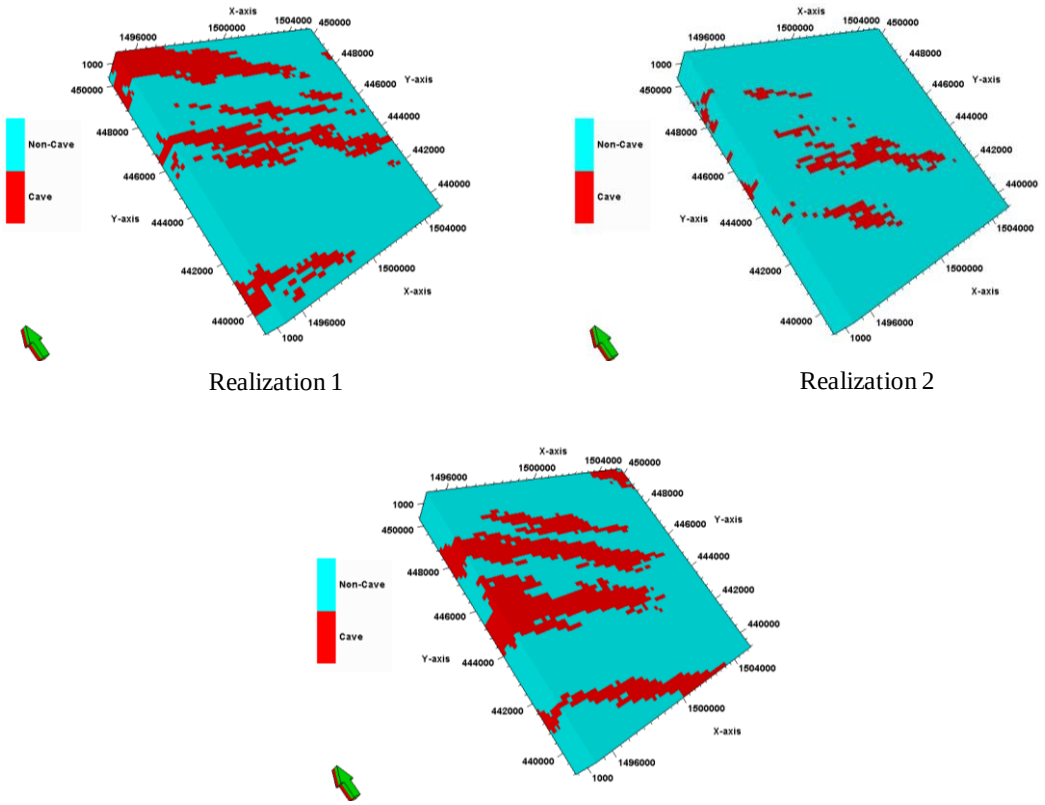
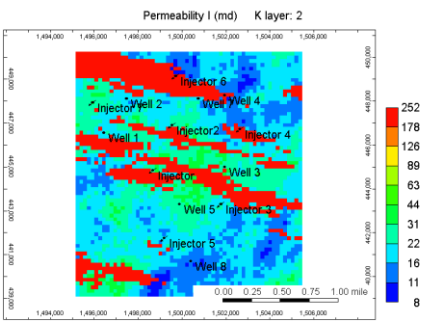
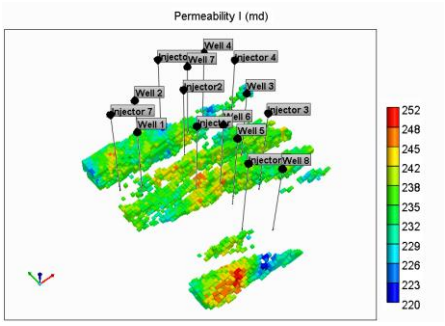


Figure 7.47- SISIM Realizations for Cave Facies. Vertical Exaggeration×3 (Red: cave facies; Cyan: non-cave facies. Green arrow shows the North direction.).

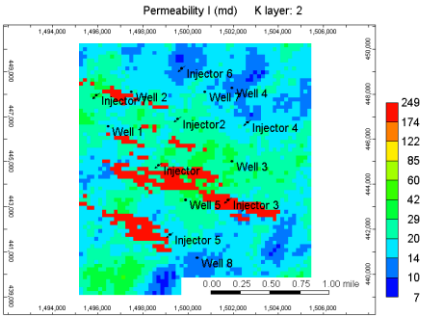
These realizations are implemented into the flow simulation model for water injection. In Figure 7.48, the distribution of cave facies in the flow simulation models and the well locations are indicated. Reservoir parameters given in Table 7.6 are also implemented and the grid thicknesses are kept the same as the model in Figure 7.36. Permeability fields given in Figure 7.34 are also used for SISIM realization models (Figure 7.48). Compared to the MPS realization 2, in SISIM realizations, cave facies are more diffused and have larger areal extent (Figure 7.48).



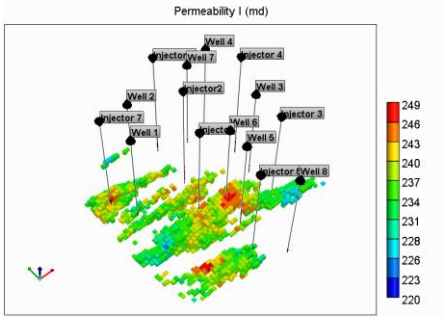
a- Realization 1 Model- Top View



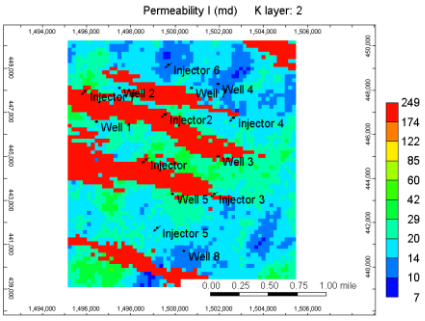
b- Realization 1 Model- Cave Facies



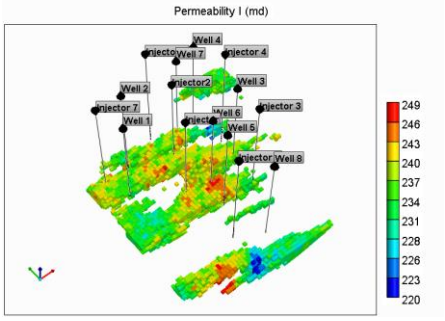
c- Realization 2 Model- Top View



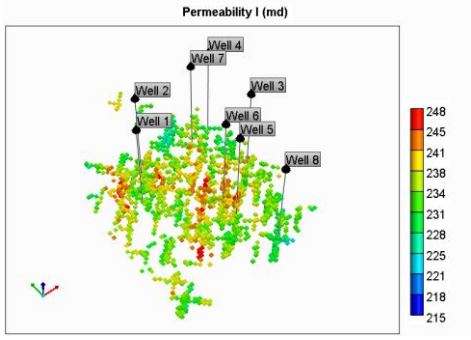
d- Realization 2 Model- Cave Facies



e- Realization 3 Model- Top View



f-Realization 3 Model- Cave Facies



g- MPS Realization 2- Cave Facies

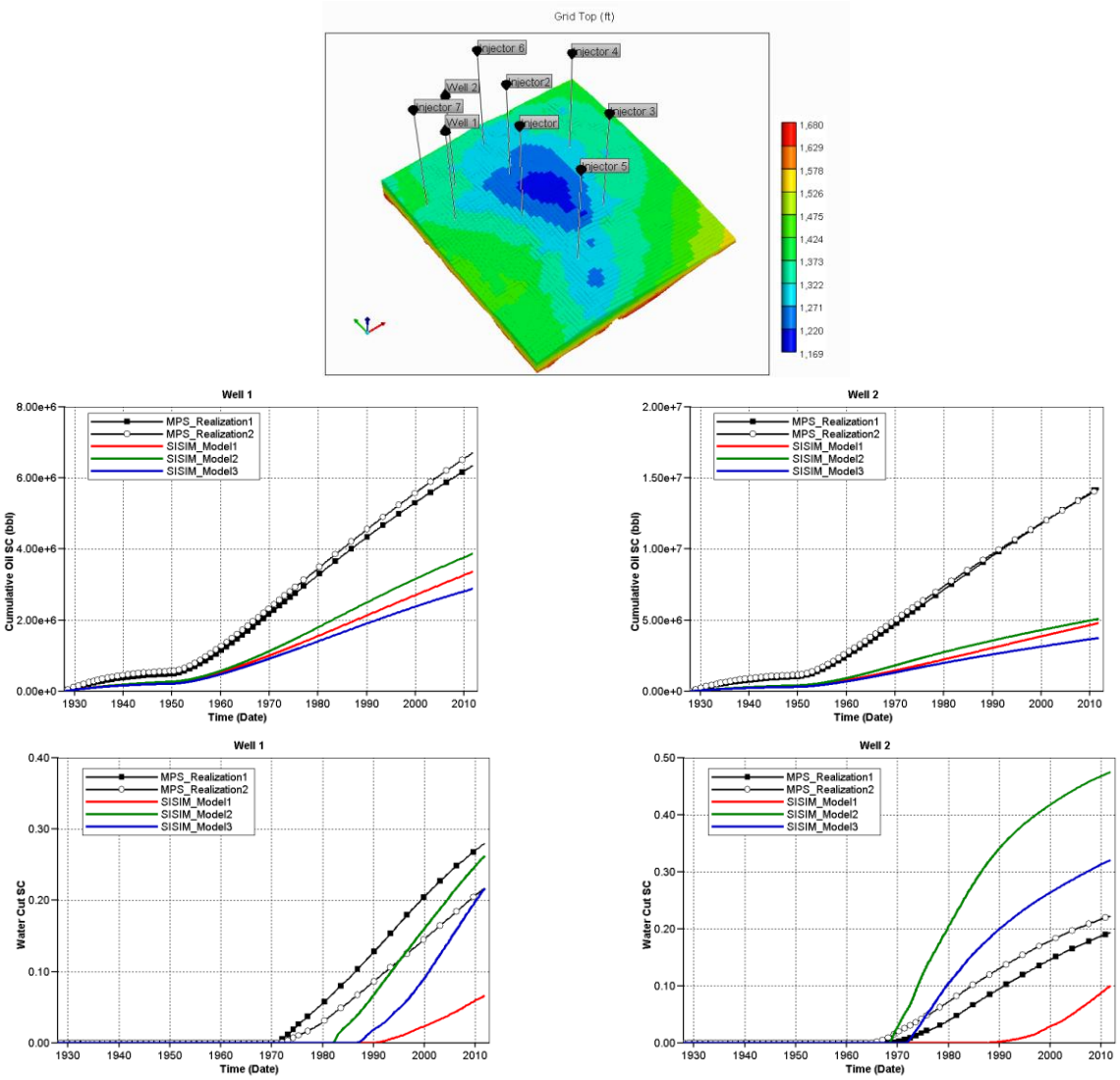
Figure 7.48- Distribution of Cave Facies Permeability in the Flow Models Constructed by SISIM. Cave facies in MPS Realization 2 is also given for comparison.

After construction of flow models for SISIM realizations, water injection is performed for the models given in Figures 7.36 (non-gridded MPS) and 7.48 (SISIM realizations). Seven injector wells are placed inside the region of interest with an injection rate of 500 bbl/day each. Since this is a synthetic case to demonstrate the effects of cave facies distribution on fluid flow responses, injector wells are placed such that water breakthrough is observed in all of the producers. Water injection starts in 1950 (23 years after the initial production) and it continues until 2012. The oil production and water cut profiles are compared for these 5 models (Figures 7.49-7.52). It is observed that the flow responses for the SISIM realization models are considerably different from those for the non-gridded MPS models (Figures 7.49-7.52).

In Figure 7.49, comparisons of cumulative oil and water cut profiles are given for Wells 1 and 2. Although production profiles have similar trends for all the models, SISIM models' cumulative oil production and water cut profiles are different than the MPS-based models. For both Wells 1 and Well 2, SISIM models yield lower oil production whereas water cut profiles have significant variations. Also, water breakthrough times are different and SISIM models have higher uncertainty. This is to be expected because the cave features are more diffused in the SISIM models and they are not described in detail.

Similar observations are obtained for Wells 3 and 4 (Figure 7.50). Range of production profiles from the different SISIM models is very wide for these wells. In Well 3, production profiles of SISIM Models 1 and 2 are in fair agreement with MPS-based models. Despite this agreement, water-cut profiles and water breakthrough times have a wider range. For Well 4, SISIM models 2 and 3 yield similar results to the MPS-models. When the cave facies distributions are compared (Figures 7.37 and 7.48), it is determined that at the Well 4 location, non-cave facies are mostly abundant in both MPS-models and SISIM models 2 and 3. On the other hand, in SISIM model 1, cave facies are mostly

dominant around Well 4 yielding extremely high production and early water breakthrough.



Well 1 Well 2

Figure 7.49- Comparison of Cumulative Oil and Water Cut Profiles for Wells 1 and 2. Well locations are also given. (Solid lines with markers: models with cave network constructed by non-gridded mps; Dashed lines: models with SISIM realizations).

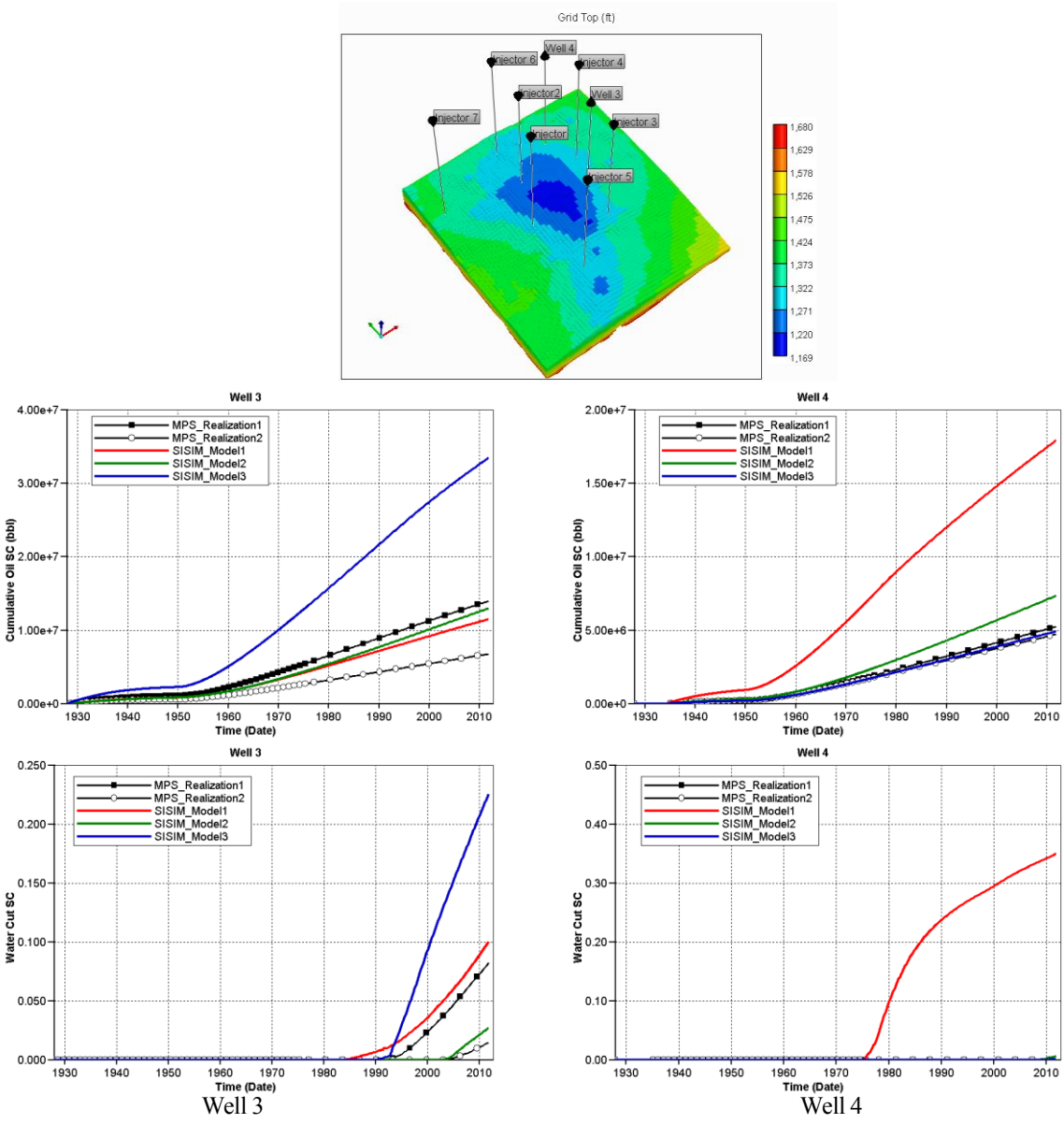
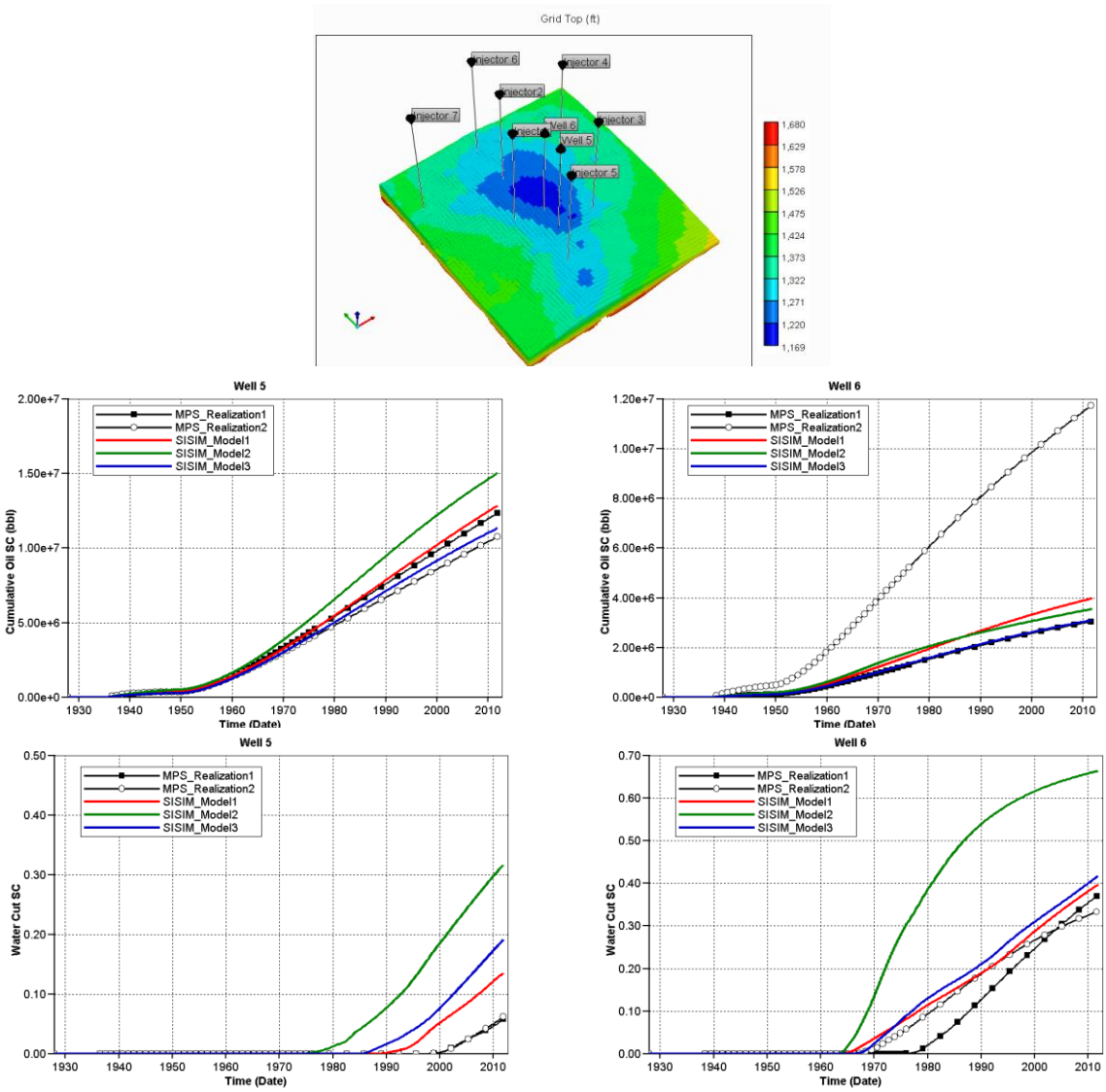


Figure 7.50- Comparison of Cumulative Oil and Water Cut Profiles for Wells 3 and 4. Well locations are also given. (Solid lines with markers: models with cave network constructed by non-gridded mps; Dashed lines: models with SISIM realizations).

Further comparisons are performed for Wells 5 and 6 (Figure 7.51). In Well 5, simulation results for SISIM models 1 and 3 are in a fair agreement with the MPS-models. On the other hand, SISIM model 2 results in higher production with higher water

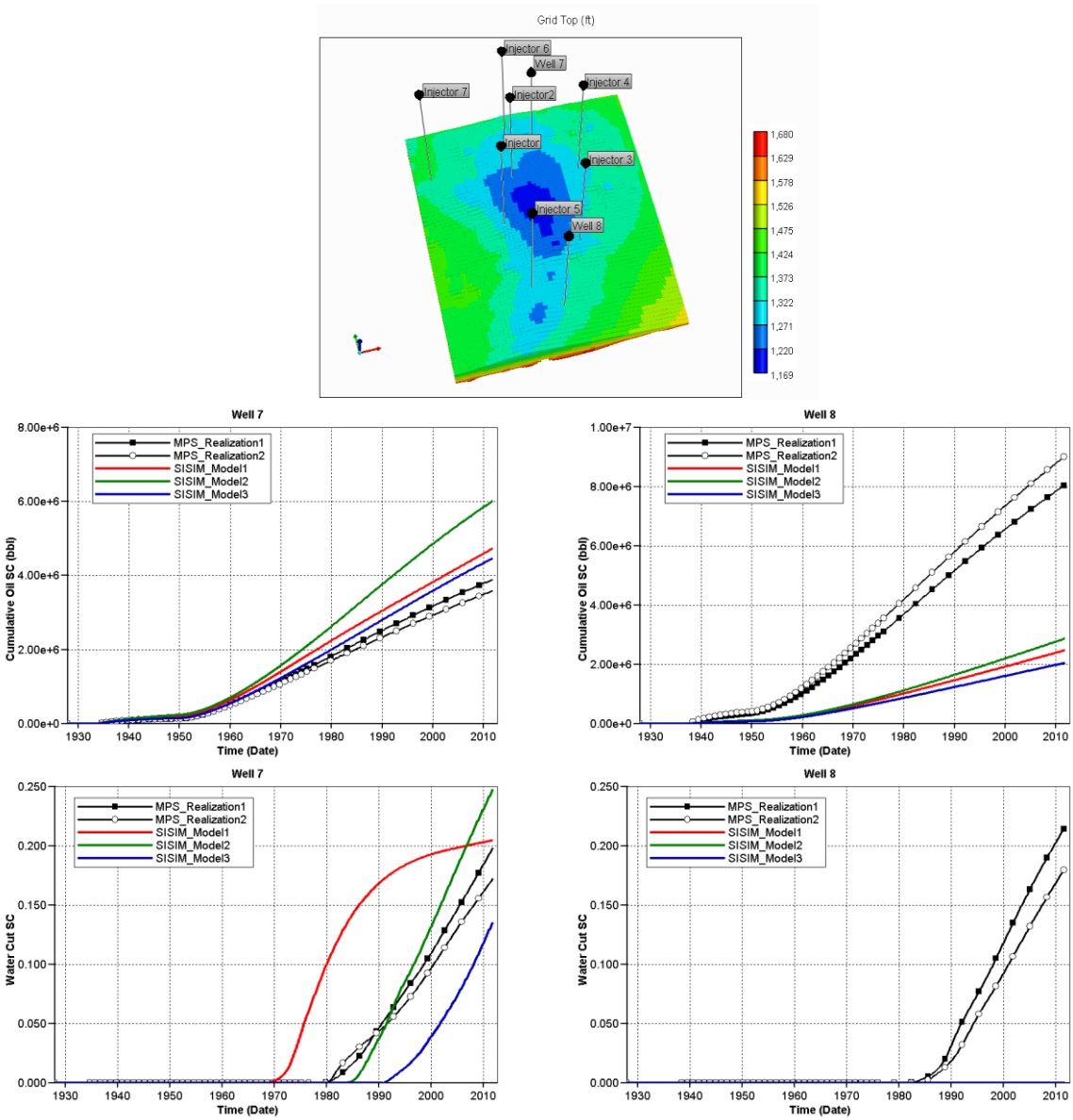
cut and early breakthrough. For Well 6, SISIM Realization models result in similar production rates compared to the MPS results. However, water cut profiles are significantly different for all models. This is mainly due to the different orientation of high permeable pathways between the injectors and Well 6 in different realizations.

Finally, cumulative oil production and WOR profiles are compared for Wells 7 and 8 (Figure 7.52), and similar conclusions are also obtained for these wells. For Well 7, SISIM realization models yield higher production rates, although all models have similar trends. Moreover, water cut profiles are different and breakthrough times show variations. In Well 8, production profiles of SISIM models are different than the MPS-based models. SISIM realizations yield notably lower production without water breakthrough. This is mainly because of the lack of cave facies (connectivity) between the injector and Well 8 (Figure 7.48) in SISIM realization models.



Well 5 Well 6

Figure 7.51- Comparison of Cumulative Oil and Water Cut Profiles for Wells 5 and 6. Well locations are also given. (Solid lines with markers: models with cave network constructed by non-gridded mps; Dashed lines: models with SISIM realizations).



Well 7 Well 8

Figure 7.52- Comparison of Cumulative Oil and Water Cut Profiles for Wells 7 and 8. Well locations are also given. (Solid lines with markers: models with cave network constructed by non-gridded mps; Dashed lines: models with SISIM realizations).

By using different flow simulation models with cave networks that are constructed by non-gridded MPS and SISIM, the effect of spatial distribution of cave

facies on the flow responses is demonstrated for a water injection case. This application yields some valuable insights regarding the use of variogram-based algorithms for modeling paleokarst reservoirs. First, it is observed that SISIM does not yield realizations in which the cave facies are simulated along a defined network structure; instead they are diffused along the main orientation defined by variogram. This situation is inconsistent with nature of the paleokarst settings which has open caves and high permeable pathways specifically along a cave network. Moreover, it is challenging to represent the complex and curvilinear structure of cave systems using two-point statistics. SISIM realizations show that variogram based modeling tools are insufficient in capturing heterogeneities in paleokarst systems.

Second, the water injection application demonstrates that detailed description of cave facies is essential in reservoir models since the reservoir connectivity drastically affect the fluid flow response. This application shows that the models constructed with SISIM result in significantly different fluid flow responses; in some wells, these models yield higher oil production (Wells 3, 4 and 7) and in some wells, oil production is significantly lower (Wells 1, 2 and 8). Also, water breakthrough times are different at most of the wells. The range of uncertainty represented by the SISIM models is therefore very wide (especially in Wells 3 and 4). In the MPS based models, the flow of water is constrained well within crisply defined cave geometries. This results in a more compact range of uncertainty.

In conclusion, the cave networks simulated by using the non-gridded MPS analysis and pattern simulation algorithms are more geologically consistent than the ones obtained by indicator simulation. Non-gridded MPS results in connected features which strictly follow a given cave orientation while having curvilinear structures. On the other hand, indicator simulation yields more diffused features without a network orientation.

Moreover, flow simulation applications show that it is essential to have a detailed description of cave facies in the reservoir to have accurate results and better future predictions. Thus, non-gridded MPS analysis and pattern simulation algorithms provide a better insight in modeling of paleokarst system.

Chapter 8: Conclusions and Suggestions Future Work

8.1 Summary of the Dissertation

Paleokarst reservoirs have complex cave networks with heterogeneous structures and it is challenging to characterize the spatial distribution of cave facies using variogram based geostatistical algorithms. Since MPS has been successfully used for modeling complex structures, such as fractured and channeled systems, paleokarst reservoirs can also be simulated by using MPS. However, analog data for inferring connectivity patterns of cave networks is unavailable in the form of gridded training images. Instead, analog data is available in the form of point-set data.

In this study, a new multiple point statistics based algorithm was developed for simulating karst networks using sparse data. The main objectives of this research as stated in Chapter 1 are;

- To infer pattern statistics regarding the spatial distribution of cave networks from modern cave analogs.
- To combine the MP-statistics together with reservoir-specific conditioning information and prior geologic knowledge to simulate the spatial distribution of cave facies.
- To demonstrate integration of simulated cave networks in flow models for accurate representation of the reservoir.

The first objective of this study is achieved by developing a unique MPS analysis algorithm to infer statistics from modern cave networks that are only represented by central line coordinates sampled along the accessible cave passages. Details of the algorithm are in Chapter 3. Unlike the conventional MPS algorithms, non-gridded MPS technique uses point-set data as training images and spatially flexible templates that are described by a group of points whose distance and angle configurations are defined with

respect to a center node. Thus, the developed technique is practical by eliminating use of grids and gridding procedure, which is challenging to apply on cave network because of its complex structure. The calibrated statistics are presented in the form of pattern histograms that report frequency (i.e. number of occurrences) of observed patterns (i.e. configurations identified by the spatial template in the point-set training image). To capture connected patterns in the training image, a set of templates with various size and configurations are implemented. Applications of non-gridded MPS analysis on various synthetic data sets are also in Chapter 3.

For the second objective, a pattern simulation algorithm is developed for modeling cave networks. Details of the algorithm are presented in Chapter 4. The calculated MP-statistics are combined with reservoir-specific conditioning information and prior geologic knowledge, to simulate the spatial distribution of cave networks. The MP-statistics inferred from the training image are used to constrain the pattern simulation and to represent connected geologic features. The simulation is conditioned to sparse data in the form of locations with cave facies or coordinates of cave structures. Cave networks are simulated by using a pattern growth-based algorithm; simulated cave passages start growing at the conditioning data locations that are visited randomly. For accurate modeling of cave network distribution, the pattern simulation algorithm provides a variety of options such as implementation of single or multiple spatial templates and a servo mechanism of histogram filtering that reduces the noise in simulation results. Moreover, by integrating quadrant frequency information of the target network, the algorithm allows network growth that is consistent with the target network or the regional geology. Furthermore, the pattern simulation algorithm integrates the dip map for the host formation to honor the local structural geology. This option allows modeling of networks that reside in formations with varying dip such as ramps and anticline structures.

The third objective in this dissertation is to integrate simulated cave networks in flow models. For this purpose, an algorithm that concurrently simulates cave openings and cave networks is developed. A demonstration on a synthetic cave opening dataset is in Chapter 5. The cave opening values are obtained by sequential Gaussian simulation that is conditioned to the cave zone thicknesses given at the conditioning locations. Cave openings are first obtained at the simulated nodes and then calculated along the simulated passages by linear interpolation between the two end points of the passage. This implementation results in a simulated cave network with cave zone thicknesses along the passages.

Once the cave networks and cave openings are simulated, a flow simulation model is constructed by implementing the simulated cave network and corresponding cave zone thicknesses. This procedure is demonstrated by an application to the Yates field of West Texas. The results are given in Chapter 7. Moreover, fluid flow responses are compared for a water injection case using several flow simulation models constructed by non-gridded MPS and sequential indicator simulation. Detailed description of spatial distribution of cave facies yields better representation of the reservoir and results in accurate fluid flow predictions. Moreover, accurate representation of reservoir connectivity features, such as cave facies, is essential since the fluid flow profiles are significantly affected by the underlying facies model. Since non-gridded MPS provides detailed description of cave facies, the corresponding reservoir models are concluded to be more representative.

In conclusion, the non-gridded MPS analysis and pattern simulation algorithms provide an insight into characterization and modeling of complex cave networks. The MPS technique developed in this research is a unique method that enables one to infer statistics from point-set data and eliminates gridding that might not be practical for

complex networks. Pattern simulation algorithm is a tool for modeling cave networks using the MP-histograms. The developed algorithm adds value to the reservoir simulation models by providing an accurate prediction for spatial distributions of cave facies in subsurface. Thus, complex reservoirs such as paleokarst systems are better represented in reservoir engineering studies and more reliable results in future predictions can be obtained.

8.2 KEY CONCLUSIONS

The followings are the major conclusions drawn from this research:

- The non-gridded MPS analysis and pattern simulation algorithms are unique techniques. For the first time, MP- statistics of point-set network data are directly calculated without using grids. MP-statistics are obtained by spatially flexible non-gridded templates whereas network orientation and data configurations are transferred by pattern histograms.
- Cave network characteristics are inferred using the non-gridded MPS analysis algorithm that uses point-set training images and spatially flexible templates. Therefore, it is essential to construct templates that inherit the overall network connectivity and continuous cave passage size. Moreover, the algorithm is MPS based and requires stationarity.
- To determine optimum template configurations, a set of generic templates are used and connections captured by each template are compared. Spurious connections might be obtained by using large templates, hence it is important to verify whether the captured patterns are consistent with the geology or not. Also, by assigning tighter tolerance windows, spurious connections can be eliminated.

- Once main orientation of the cave network is determined, complete spatial templates are constructed and MP-histograms are obtained by scanning the point-set training image. The resultant pattern histograms transfer the network connectivity information as well as cave passage extents.
- Pattern simulation of cave network is conditioned to sparse data in the form of locations with cave facies or coordinates of cave structures. Thus, in paleokarst reservoirs, it is important to identify wells with cave facies and corresponding cave zone thicknesses to simulate the cave network. Also, it is essential to use MP-statistics inferred from training images that are consistent with the reservoir geology.
- Cave network simulation results are improved by using the options available in the algorithm. Noise in the simulated network is reduced by eliminating the simulated patterns which are not observable in the target image. Moreover, using quadrant frequency information of the target network and implementing host formation dip maps result in accurate simulation results.
- Uncertainty in the simulated cave networks can be assessed by using different templates with various configurations and by varying the tolerance window used in the pattern simulation.
- Using single or multiple templates results in simulated cave networks with different passage configurations. Therefore, it is important to assign templates that represent the target cave network structure.
- Cave facies in paleokarst reservoirs can be identified using GR, bulk density and caliper logs. Criteria for cave facies identification are defined by GR<40 API, bulk density<2.3 g/cc and 1.5 in caliper log deviation from a caliper baseline. Thus, distribution of cave facies in a small region of the Yates Field

is determined using well logs. Identified cave facies intervals are compared to the core descriptions. The proposed criteria are successful in prediction of cave facies. Also, calculated distribution of cave zone heights are consistent with the ones that are previously reported.

- Simulated cave networks and cave opening data are integrated in a flow simulation model. An application is presented for the Yates Field. By implementing flow models constructed by non-gridded MPS and sequential indicator simulation for a water injection demonstration, it is determined that reservoir connectivity significantly affects the fluid flow responses and incorrect representation of cave network connectivity can result in inaccurate predictions. It is concluded that detailed description of connectivity features such as cave facies is essential for accurate reservoir representation and better flow predictions.
- In conclusion, the proposed algorithms yield accurate description of cave networks and paleokarst reservoirs. The non-gridded MPS analysis and simulation algorithm applies to any kind of network data that is defined by point-sets. Pattern simulation using the calibrated statistics is a practical to tool to model spatial distribution of cave facies. Moreover, these techniques are practical and eliminate the gridding procedure, which could be challenging for complex networks.

8.3 Recommendations for Future Work

Based on the results and the conclusions drawn from this dissertation, recommendations for further applications of the developed algorithms are given below:

- In this dissertation, non-gridded MPS analysis and pattern simulation algorithms are applied on epigenic type of cave networks. Further applications

of these methods on hypogenic type of karst systems can be performed for comparison of the inferred MP-statistics. Distinction in statistics of epigenic and hypogenic cave networks can be useful in modeling karstic reservoirs with different formation origins.

- Although an application of the developed techniques on a fault network is presented in this study, detailed analysis and further applications can be conducted for fracture networks. Thus, fracture networks in reservoirs can be simulated practically without using grid-based algorithms. Once the main fracture orientations are identified, fracture network can be simulated using the FMI logs or any kind of fracture image data. The images must be digitized to construct a point-set that can be implemented as a non-gridded training image.
- Uncertainty in the simulated cave networks can be assessed by implementing various spatial templates and by using different tolerance windows. Therefore, it is worthwhile to perform further analysis to determine the effects of template structure and tolerance window size on the simulated networks and fluid flow responses.
- Spatial templates do not use any information regarding to the node connections while scanning the image, instead node configurations and tolerance values are considered only. Therefore, for further improvements, non-gridded training images can be implemented in the form of both network points and corresponding connections, similar to a graph structure. Thus, spatially flexible templates will consider both configuration and connection relations while scanning the training image and constructing the pattern histogram.

- There are recent attempts in the literature to introduce non-stationarity in conventional grid-based MPS algorithm. Similarly, applications of the non-gridded MPS analysis on non-stationary domains can be examined. A multi-grid approach can be followed by implementing templates at different scales to calculate statistics of non-stationary training images.
- It is worthwhile to conduct studies on identification of cave facies. Although the proposed criteria are successful in recognition cave facies in this study, the identification is purely based on well log data. Thus, any uncertainty or artifacts in well logs may result in incorrect description. Therefore, further analyses in recognition of cave facies can be performed by integrating various data sources, such as FMI and neutron logs, well tests, micro seismic data and thin section analysis of core samples. Moreover, Capacitance-Resistance Models (CRM) can be used to identify connected geologic structures and spatial templates can be constructed based on CRM results.
- Further studies can be conducted to develop a method for history matching analysis in paleokarst reservoirs. By integrating the simulated cave network and petrophysical properties, a robust framework for efficient and practical history matching analysis in complex reservoirs can be devised.
- Applicability of non-gridded MPS analysis and pattern simulation can be tested on reservoirs with connected geologic features such as channels. Once the channel orientations are determined by using geology, seismic data and well logs, locations with identified channel sediments can be determined and used in non-gridded MPS analysis.
- In this dissertation, an algorithm is developed for modeling cave width and cave opening. It is used for simulating the cave facies thicknesses in the Yates

Field. This algorithm can also be applied on modeling of cavern space for a cave system with measured cave opening data. Thus, 3D volumetric cave models can be obtained using this algorithm and the results can be compared with measured 3D cave surveys.

- The cave network simulation algorithm can also be applied in hydrology studies for modeling of aquifers and groundwater flow. The network simulation procedure will be the same for aquifer modeling. Since in hydrology applications, groundwater flow measurement stations are common and frequently spaced, conditioning data locations will be exhaustive and corresponding analysis will be informative.
- Probability field (p-field) representing the spatial distribution of host rock formation of the cave system can be integrated as a secondary data into the network simulation algorithm. This application is similar to the implementation of quadrant frequency data; if the p-field of the host rock is available, then the cave network will preferentially grow from locations with higher probability. The p-field map can be constructed by integrating various information on the host rock, such as thickness variation of the host formation, presence of any discharge locations (for hydrological applications) and exposed cave features such as outcrops.

Appendix A: Non-Gridded MPS Analysis Code

A.1 Overview

Non-gridded MPS analysis algorithm is developed in C++ using Microsoft Visual Studio 2008. The details and applications of the algorithm are given in Chapter 3. In Figure A.1, a point-set training image file is shown. The training image consists of XYZ coordinates of the data points.

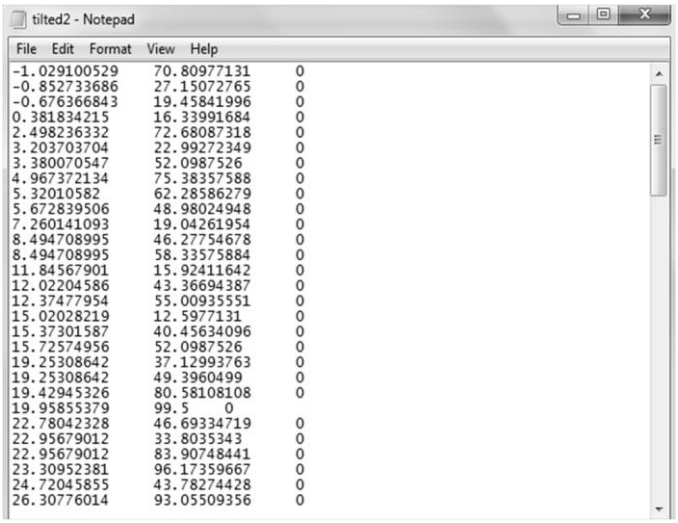


Figure A.1- Sample Point-Set Data File. The columns are x, y and z coordinates.

Non-gridded MPS analysis is performed by using spatially flexible template which is defined as by the configuration of template nodes around a center point. A sample file including template description is given in Figure A.2. The first line should give the number of template nodes (excluding the center node). Consecutive lines describe template node configurations given by azimuth angle, dip angle and lag distance respectively. Angle and distance tolerance values are currently defined within the source code; however, these values could be directly read from an input file or via the command window.

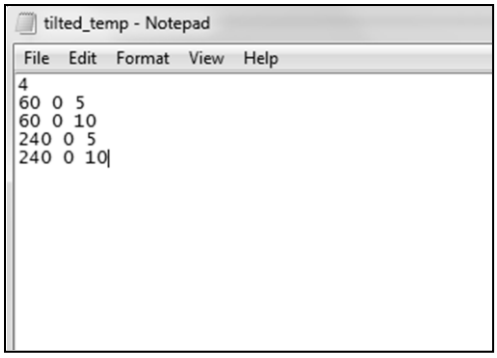


Figure A.2- Sample Template File. First line is the number of template nodes. Template nodes are given after the second line; the first column is azimuth angle, the second column is dip angle and the third one is lag distance.

The point-set training image is then scanned using the non-gridded spatially flexible template. Input files are given via the command window when the non-gridded MPS analysis program is executed (Figure A.3).

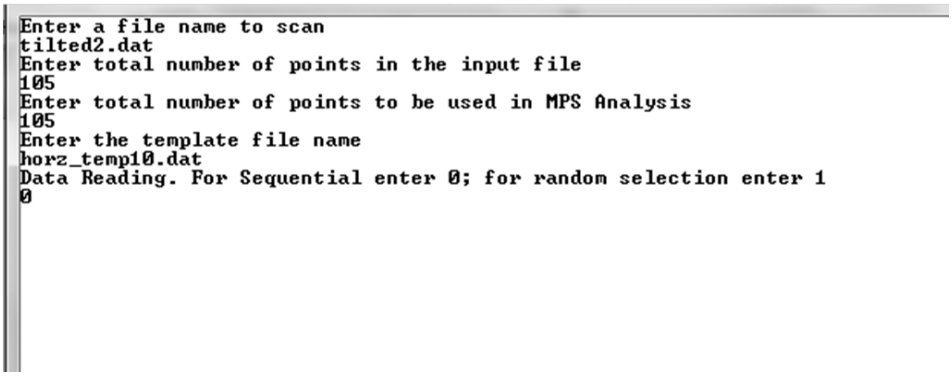


Figure A.3- Sample Command Window Input.

Output files of the non-gridded MPS analysis are a pattern histogram, captured connections (one file in Petrel general line format, one file with line sticks and one file with connections in node number), template nodes of the observed patterns, decimal numbers representing the patterns, list of connected node coordinates and file reporting total number of captured connections.

A sample pattern histogram file with 6 patterns is given in Figure A.4. First column is pattern number corresponding to the pattern's order in tempnodes file (Figure A.5). Second column is pattern frequency whereas the third column gives pattern length (number of nodes in the configuration). Fourth column is decimal number representing the pattern and this number is unique for each pattern. These decimal numbers are also stored in as an output file.

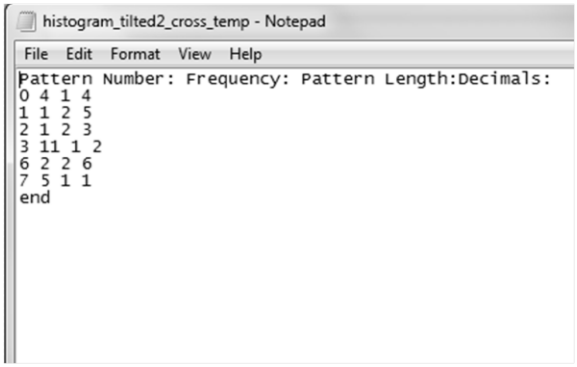


Figure A.4- Sample Pattern Histogram File- Output of Non-Gridded MPS. Column names are given in the first line. Decimals represent the pattern.

Besides the pattern histogram, template node and decimal files, non-gridded MPS analysis yields additional files including captured connections in different formats. The results can be directly visualized as general lines in Petrel (Schlumberger, 2010) with "line_connections_petrel___.dat" file. Moreover, captured connections are also given as a general line stick format in "connections__.dat" file and connected node numbers are given in "nodes___.dat". The node numbers are associated with the connected node list stored in "locations__.dat" file.

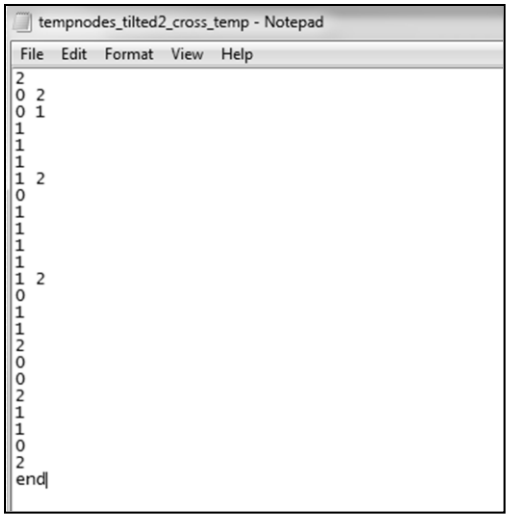


Figure A.5- Sample Tempnodes File- Output of Non-Gridded MPS. Each row shows the pattern observed at a particular data location. Template nodes that are "active" in the pattern are given. Node indexing starts from 0, i.e. first template node is 0.

A.2 HEADER FILE : MPS_RAD_HEADER.H

```
#ifndef MPS_rad_header
#define MPS_rad_header

using namespace std;
class RetInd {
  public:
      vector <int> Ta;
      vector <int> Tb;
};
//=========================================================//
//Converts degrees to radians
double deg2rad(double Deg) {
      double pi_a= 3.14159;
      double Rad=(Deg*pi_a)/180;
      return Rad;
}

//Calculate node coordinates based on the given configuration
double   *BackCal(double   AzmAng,double   DipAng,double   SepDist,double
X1,double Y1,double Z1)
{     double Y2,Dist1,X2,Z2;
      X2=X1;
      Y2=Y1;
      Z2=Z1;

      Dist1=SepDist*sin(AzmAng);
      X2=X1+Dist1*cos(DipAng);
      Y2=Y1+SepDist*cos(AzmAng);
```

```
	Z2=Z1+Dist1*sin(DipAng);
	double *Coord2=new double[3];
	Coord2[0]=X2;
	Coord2[1]=Y2;
	Coord2[2]=Z2;
	return Coord2;
}

//Determine Maximum and Minimum X,Y,Z coordinates of Cave Data
vector <double> bounds(double **Array,int Select, double numD) {
//Array = Data
	vector <double> Vals (3);   //[Xmin, Ymin, Zmin] //[Xmax, Ymax,
Zmax]

Vals.at(0)=Array[0][0];
Vals.at(1)=Array[0][1];
Vals.at(2)=Array[0][2];
//Select=1 for minimum, Select=2 for maximum
switch (Select) {
	case 1: //Determine Minimum
		for (int t=1; t<numD; t++) {
			//Minimum X Coordinate
			if (Array[t][0]<Vals.at(0)) {
			Vals.at(0)=Array[t][0];
			}
			//Minimum Y Coordinate
			if (Array[t][1]<Vals.at(1)) {
			Vals.at(1)=Array[t][1];
			}
			//Minimum Z Coordinate
			if (Array[t][2]<Vals.at(2)) {
			Vals.at(2)=Array[t][2];
			}

		}
		break;

	case 2: //Determine Maximum
		for (int t=1; t<numD; t++) {
			//Maximum X Coordinate
			if (Array[t][0]>Vals.at(0)) {
			Vals.at(0)=Array[t][0];
			}
			//Maximum Y Coordinate
			if (Array[t][1]>Vals.at(1)) {
			Vals.at(1)=Array[t][1];
			}
			//Maximum Z Coordinate
			if (Array[t][2]>Vals.at(2)) {
			Vals.at(2)=Array[t][2];
			}
		}
		break;
	} //End of switch
```

```
return Vals;
}

//****Boundary Control**//
int  BndContl(vector<double>Node,vector<double>MinB,vector<double>  MaxB)
{
      //BndCond=0: Outside the boundary; BndCond=1: Inside the boundary
      int BndCond=0;

      if (Node[0]>=MinB.at(0) && Node[0]<=MaxB.at(0)) {
            if (Node[1]>=MinB.at(1) && Node[1]<=MaxB.at(1)) {
                  if (Node[2]>=MinB.at(2) && Node[2]<=MaxB.at(2)) {
                        BndCond=1;
                  }
                  else {BndCond=0;}
            }
            else {BndCond=0;}
      }
      else {BndCond=0;}

      return BndCond;
}

//Find whether an element is in the vector or not
int findE(int vec1,vector <int> vec2) {
      //cond=1 : element is in the vector. cond=0 : element is not in
the vector
      int cond=0;
      int cts=0;
      for (int kl=0; kl<vec2.size(); kl++) {
                  if (vec1==vec2.at(kl)) {
                        cts++;
                        if (cts>0) {
                        cond=1;
                        break; }
                  }
            }
      return cond;
}

//Find indices of elements for the given condition for integers
vector<int> findanyI(vector <int> Vec, int MyVal, int MyCond) {
//MyCond=0 Equal to; MyCond=1; Smaller than; MyCond=2; Greater than
      int kt;
      vector <int> km;

      for (kt=0; kt<Vec.size();kt++) {
            if (MyCond==0) {
                  if (Vec.at(kt)==MyVal) {
                        km.push_back(kt); }
            }
            if (MyCond==1) {
                  if (Vec.at(kt)<=MyVal) {
                        km.push_back(kt); }
```

```
            }

            if (MyCond==2) {
                  if (Vec.at(kt)>=MyVal) {
                        km.push_back(kt); }
            }
      }

      return km;}

//**Calculation of Difference**//
double Delta(double Loc1,double Loc2){
      double dif;
      dif=fabs(Loc1-Loc2);
      return dif;}

//**Calculation of Lag Distance**//
double Cal_h(double Loc1X,double Loc1Y,double Loc1Z,double Loc2X,double
Loc2Y,double Loc2Z){
      double CalH;
      double DX=Delta(Loc1X,Loc2X);
      double DY=Delta(Loc1Y,Loc2Y);
      double DZ=Delta(Loc1Z,Loc2Z);

      CalH=sqrt(pow(DX,2)+pow(DY,2)+pow(DZ,2));
      return CalH;}

//**Calculation of Distance (2D)**//
double Cal_d(double Dis1,double Dis2){
      double Dis3;
      Dis3=sqrt(pow(Dis1,2)+pow(Dis2,2));
      return Dis3;}

//***Find array indices fall into the tolerance window of template
nodes***//
vector <int> find_temp(double dist,double dist_tol_f,double rad,double
rad_tol,double rdip,double dip_tol, vector <double> vec_dist, vector
<double> vec_ang,vector <double> vec_dip) {
      double dist_tol;
      double pi_a= 3.14159;
      vector <int> cond;
            for (int kl=0; kl<vec_dist.size(); kl++) {
                  dist_tol=dist*dist_tol_f;
                  if (dist-dist_tol/2<=vec_dist.at(kl) &&
vec_dist.at(kl)<=dist+dist_tol/2 && rad-rad_tol/2<=vec_ang.at(kl) &&
vec_ang.at(kl)<=rad+rad_tol/2 && rdip-dip_tol/2<=vec_dip.at(kl) &&
vec_dip.at(kl)<=rdip+dip_tol/2) {
                              cond.push_back(kl);
                        }

                  }
      return cond;
}
```

```
//***Scan Data Configuration ***//
RetInd search_temp(vector< vector<double>> CTemp,int sz,vector<double>
LagAr,vector<double> AzmAr, vector <double> DipAr,double DistTol_F,
double AzmTol, double DipTol) {     //sz:Template size

      vector <int>Indx(0);
      vector<int>Tempo;
      vector <int> TempInd;
      double pi_a= 3.14159;

      int tl,rnd;
      for (tl=0; tl<sz; tl++) { //Loop over Template nodes

            if (CTemp[tl][0]<0) {
                  CTemp[tl][0]=2*pi_a-fabs(CTemp[tl][0]);
            }

      Tempo=find_temp(CTemp[tl][2],DistTol_F,CTemp[tl][0],AzmTol,CTemp[
tl][1],DipTol,LagAr,AzmAr,DipAr);

            if (Tempo.empty()!=1) {

                  rnd=0;

                  if (Tempo.size()>1) {
                        rnd= rand() %Tempo.size();
                  }

                        if (findE(Tempo.at(rnd),Indx)==0) {
                              TempInd.push_back(tl);
                              Indx.push_back(Tempo.at(rnd));

                        }
            Tempo.clear();
            }

      }//End of Loop over template nodes: for (tl=0; tl<sz; tl++)

      RetInd ArrRet;
      ArrRet.Ta=Indx;
      ArrRet.Tb=TempInd;

      return ArrRet;
}

//Calculates Azimuth Angle
double    Cal_Azm(double    Loc1X,double    Loc1Y,double    Loc1Z,double
Loc2X,double Loc2Y,double Loc2Z){
      double Azm_Ang=0;
      double Dist=Cal_d(fabs(Loc1X-Loc2X),fabs(Loc1Z-Loc2Z));
      double LagD=Cal_h(Loc1X,Loc1Y,Loc1Z,Loc2X,Loc2Y,Loc2Z);
      double pi_a= 3.14159;

      if (Loc1Y!=Loc2Y){
```

```
            if(Loc2Y>Loc1Y) {
                  if (Loc2X>Loc1X) {
                        Azm_Ang=atan(Dist/(Loc2Y-Loc1Y));
                  }
                  if (Loc2X<Loc1X) {
                        Azm_Ang=2*pi_a-fabs(atan(Dist/(Loc2Y-Loc1Y)));

                  }

                  if (Loc2X==Loc1X) {
                  Azm_Ang=0;
                  }
            }
            else {
                  if (Loc2X>Loc1X) {
                        Azm_Ang=pi_a-fabs(atan(Dist/(Loc2Y-Loc1Y)));
                  }

                  if (Loc2X<Loc1X) {
                        Azm_Ang=pi_a+fabs(atan(Dist/(Loc2Y-Loc1Y)));

                  }

                  if (Loc2X==Loc1X) {
                  Azm_Ang=pi_a;
                  }
            }
      }
      else {
            if(Loc2X==Loc1X) {
                  if(Loc2Z>Loc1Z) {
                  Azm_Ang=pi_a/2;
                  }
                  if(Loc2Z<Loc1Z) {
                  Azm_Ang=3*pi_a/2;
                  }
            }
            else {
                  if(Loc2X>Loc1X) {
                  Azm_Ang=pi_a/2;
                  }

                  if(Loc2X<Loc1X) {
                  Azm_Ang=3*pi_a/2;
                  }
            }
      }
      return Azm_Ang;
}

//Calculates Dip Angle
double    Cal_Dip(double    Loc1X,double    Loc1Y,double    Loc1Z,double
Loc2X,double Loc2Y,double Loc2Z){
```

```
      double Dip_Ang;
      double Dist=Cal_d(fabs(Loc1X-Loc2X),fabs(Loc1Z-Loc2Z));
      double DelZ=Delta(Loc1Z,Loc2Z);
      double pi_a= 3.14159;

      if (Loc1Z!=Loc2Z) {
            Dip_Ang=(atan((Loc2Z-Loc1Z)/(Loc2X-Loc1X)));//Dip angle

      }

      else { Dip_Ang=0; }

      return Dip_Ang;
}

//Converts integer to string
string intToString(int t){
      std::string ch;
      ostringstream outs;
      outs << t; // Convert value into a string.
    ch = outs.str();
      return ch;
}

//Extracting Frequencies by using Decimal Data***//
int GetFreq(int Data,vector <int>DataVec) {
      int F=0;
      for (int s=0; s<DataVec.size(); s++) {
            if (Data==DataVec.at(s)) {
                  F++;
                  }

      }
      return F;
}
//=============================================================//
#endif // MPS_rad_header
```

A.3 NON-GRIDDED MPS ANALYSIS ALGORITHM

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <iomanip>
#include <cmath>
#include <vector>
#include <bitset>
#include <algorithm>
#include <string>
#include "MPS_rad_header.h"

using namespace std;
```

```
int nrow; //Size of input file (number of rows)
int n; //Number of data to be used in MPS analysis. If there is a
computation limitation, n could be smaller than nrow.

int i,j, ctt, ReadType;
double Template[27][3];
int* Index= new int [];

double Azm_Tol=10; //Define azimuth tolerance;
double Dip_Tol=10; //Define dip tolerance;
double Dist_Tol_F=0.333; //Define lag tolerance factor
double pi = 3.14159;
int BinTDec;

ifstream infile;
ifstream infile3;

//********Output files************//
ofstream outfile; //("nodes___.dat"); //Also update the file name in
Line 327, ifstream2
ofstream outfile2;//("connections___.dat"); // Line stick format
ofstream outfile3; //Brief results of MPS analysis; total number of
connections
ofstream outfile4;//("tempnodes___.dat");
ofstream outfile5; //("decimals___.dat");
ofstream outfile6; //("locations___.dat");
ofstream outfile7; //("histogram___.dat");
ofstream outfile8; //("line_connections_petrel___.txt");(Store captured
connections in general line format of Petre)

int main()
{
      std::string ifile;
      //Input file including the point set data. (X Y Z coordinates)
      cout << "Enter a file name to scan" << endl;
      cin >> ifile;

      infile.open(ifile.c_str());
      if (!infile)
      {
      cout << ifile + "was not found.Please check the directory or
enter the file name again";
      return EXIT_FAILURE;
      }

      //Input for size of the data file and number of points to be used
in MPS Analysis
      cout << "Enter total number of points in the input file" << endl;
      cin >> nrow;

      cout << "Enter total number of points to be used in MPS Analysis"
<< endl;
      cin >> n;
```

```
      //Enter the file name for the template. It contains template
configuration
      std::string ifile2;
      cout << "Enter the template file name" << endl;
      cin >> ifile2;

      infile3.open(ifile2.c_str());
      if (!infile3)
      {
      cout << ifile2 + "was not found.Please check the directory or
enter the file name again";
      return EXIT_FAILURE;
      }

      cout<<"Data Reading. For Sequential enter 0; for random selection
enter 1"<<endl;
      cin>>ReadType;

      //==============================//
      double *XLoc= new double [n];
      double *YLoc= new double [n];
      double *ZLoc= new double [n];
      double **DelX;
      double **DelY;
      double **DelZ;
      double **D1;
      double **Phi;
      double **h;
      double **Theta;
      double **Data;

      Data=new double*[nrow];
      DelX= new double*[n];
      DelY= new double*[n];
      DelZ= new double*[n];
      D1= new double*[n];
      Phi= new double*[n];
      h= new double*[n];
      Theta= new double*[n];

      static const unsigned BIT_LENGTH = 27;
            for(i = 0; i<n; i++)
            {
            DelX[i]  =  new  double[n];//each  array  index  pointer  now
points to an array of doubles
            DelY[i] = new double[n];
            DelZ[i] = new double[n];
            D1[i] = new double[n];
            h[i] = new double[n];
            Phi[i]= new double[n];
            Theta[i] = new double[n];
            }
```

```
            for (i=0; i<nrow; i++) { //size of input file (number of
rows)
                  Data[i]=new double[3];

                  for (j=0; j<3; j++) {
                  infile>>Data[i][j];
                  }

            }

      //Load template file
      int th; //Size of the template
      infile3>>th;
      vector<vector<double>>CondTemp(th,3);
      for (i=0; i<th; i++) {
      infile3>>CondTemp[i][0];
      CondTemp[i][0]=deg2rad(CondTemp[i][0]); //Azimuth angle
      infile3>>CondTemp[i][1];
      CondTemp[i][1]=deg2rad(CondTemp[i][1]); //Dip angle
      infile3>>CondTemp[i][2]; //Distance
      }

      size_t found1;
      string str2, str3;

      found1=ifile.find(".dat");
      str2=ifile.substr(0,found1);

      found1=ifile2.find(".dat");
      str3=ifile2.substr(0,found1);

      //This file will include the data locations used in MPS analysis
      std::string name =("locations_"+ str2 +"_"+ str3 + ".dat");
      outfile6.open(name.c_str());

      int low=0, high=nrow-1;
      vector <int> rand_loc; //(n);
      int fr,rnv;
      if (ReadType==1) {
            for (i=0; i<n; i++) {

            //======== Data is randomly selected===//
                  rnv=low+((high-low+1)*rand()/(RAND_MAX + 1.0));
                        fr=findE(rnv,rand_loc);

                        if (i!=0 && fr==1) {
                              while (findE(rnv,rand_loc)==1) {
                                          rnv=low+((high-low+1) *rand()
                                          /(RAND_MAX + 1.0));
                              }
                        }
                  rand_loc.push_back(rnv);

                  XLoc[i]=Data[rand_loc.at(i)][0];
```

```
            YLoc[i]=Data[rand_loc.at(i)][1];
            ZLoc[i]=Data[rand_loc.at(i)][2];
      //=========================//

            outfile6<<XLoc[i]<<" "
                    <<YLoc[i]<<" "
                    <<ZLoc[i]<<endl;
      }
}

if (ReadType==0) {
      for (i=0; i<n; i++) {

            //====Data is imported sequentially=====
            XLoc[i]=Data[i][0];
            YLoc[i]=Data[i][1];
            ZLoc[i]=Data[i][2];

            outfile6<<XLoc[i]<<" "
         <<YLoc[i]<<" "
         <<ZLoc[i]<<endl;
      }
}

vector<double> LagD;
vector<double> Temp_Azm; //Calculated Azimuth angle
vector<double> Temp_Dip; //Calculated Dip angle
vector<int> Indx_h;
vector<int> JIndx; //J indices
vector<int> Indx_Azm;

//=================Output Files==================//
name =("nodes_"+ str2 +"_"+ str3 + ".dat");
outfile.open(name.c_str());

name =("connections_"+ str2  + "_"+ str3 + ".dat");
outfile2.open(name.c_str());

name =("total connection number"+ str2  + "_"+ str3+ ".dat");
outfile3.open(name.c_str());

name =("tempnodes_"+ str2  + "_"+ str3 + ".dat");
outfile4.open(name.c_str());

name =("decimals_"+ str2  + "_"+ str3 + ".dat");
outfile5.open(name.c_str());

name =("histogram_"+ str2  + "_"+ str3 + ".dat");
outfile7.open(name.c_str());

name =("line_connections_petrel_"+ str2  + "_"+ str3 + ".txt");
outfile8.open(name.c_str());
//=====================================================//
```

```
      int j,count, ncon; //count:number of connections
      count=0;
      ncon=0;
      double azmr;
      RetInd Arr;

      //=============================================//
      double fnum=0;
      vector <int> Decimals; //stores decimals of the patterns
      vector <int> PatLeng; //stores pattern length
       //=============================================//

      vector <double> MinBndCoor (3); //[X,Y,Z]; Minimum Boundary
      Coordinates
      vector <double> MaxBndCoor (3); //[X,Y,Z]; Maximum Boundary
      Coordinates

      //(Array,Select) Select=1 for mins, Select=2 for maxs.
      MinBndCoor=bounds(Data,1,nrow);
      MaxBndCoor=bounds(Data,2,nrow);

      vector <double> TempCord2 (3);
      double* TempCord;
      vector<int> CntlVal;
      vector <int> ValCheck;
      //**==================Analysis Starts=====================**//
      for (i=0; i<n; i++) {

            for (j=0; j<CondTemp.size(); j++) {

            TempCord=BackCal(CondTemp.at(j).at(0),CondTemp.at(j).at(1),
            CondTemp.at(j).at(2),XLoc[i],YLoc[i],ZLoc[i]);
            TempCord2.at(0)=TempCord[0];
            TempCord2.at(1)=TempCord[1];
            TempCord2.at(2)=TempCord[2];
            CntlVal.push_back(BndContl(TempCord2,MinBndCoor,
MaxBndCoor)); //0:Outside the boundary; 1:Inside the boundary
            }

            ValCheck=findanyI(CntlVal,0,0);  //Find  any  template  nodes
outside the boundary
            if (ValCheck.empty()==1 || ValCheck.size()<CntlVal.size()){
                  for (j=0; j<n; j++) {
                        if (j!=i) {
                        LagD.push_back(Cal_h(XLoc[i],YLoc[i],ZLoc[i],
                        XLoc[j],YLoc[j],ZLoc[j]));
                        JIndx.push_back(j);
                        Temp_Azm.push_back(Cal_Azm(XLoc[i],YLoc[i],
                        ZLoc[i],XLoc[j],YLoc[j],ZLoc[j]));
                        Temp_Dip.push_back(Cal_Dip(XLoc[i],YLoc[i],
                        ZLoc[i],XLoc[j],YLoc[j],ZLoc[j]));
                        }
                  }
```

```
            Arr=search_temp(CondTemp,th,LagD,Temp_Azm,
            Temp_Dip,Dist_Tol_F,deg2rad(Azm_Tol),
            deg2rad(Dip_Tol));
            Indx_h=Arr.Ta;

//Converts Observed pattern into Decimal (to get frequency)//
            //Constructs the histogram//
            for (int ct=0; ct<Arr.Tb.size(); ct++) {
                  fnum=fnum+pow(double (2),Arr.Tb.at(ct));

                  outfile4<<Arr.Tb.at(ct)<<" ";

                  if (Arr.Tb.size()-ct==1) {
                        outfile4<<endl;
                        Decimals.push_back(fnum);
                        PatLeng.push_back(Arr.Tb.size());
                        outfile5<<fnum<<endl;
                  }

            }
                  fnum=0;

                  if (Indx_h.empty()!=1) {
                        outfile<<i<<" ";

                        //Store the connection information
                        for (int ct=0; ct<Indx_h.size(); ct++) {
                        outfile<<JIndx.at(Indx_h.at(ct))<<" ";
                        //Store in connections file
                        outfile2<<ncon<<" "
                                <<XLoc[i]<<" "
                                <<YLoc[i]<<" "
                                <<ZLoc[i]<<endl;

                        //Store in connections file
                        outfile2<<ncon<<" "
                              <<XLoc[JIndx.at(Indx_h.at(ct))]
                              <<" "<<YLoc[JIndx.at(Indx_h.at(ct))]
                           <<" "<<ZLoc[JIndx.at(Indx_h.at(ct))]<<endl;
                        outfile8<<XLoc[i]<<" "<<YLoc[i]<<" "
                                    <<ZLoc[i]<<endl;
                        outfile8<<XLoc[JIndx.at(Indx_h.at(ct))]<<"
                               "<<YLoc[JIndx.at(Indx_h.at(ct))]<<"
                           "<<ZLoc[JIndx.at(Indx_h.at(ct))]<<endl;
                                    outfile8<<-999<<" "<<-999<<" "
                                                  <<-999<<endl;
                                                count=count++;
                                                      ncon++;
                        }

                  outfile<<endl;

                  }//End of if (Indx_Azm.empty()!=1)
```

```
                        LagD.clear();
                        JIndx.clear();
                        Indx_h.clear();
                        Temp_Azm.clear();
                        Temp_Dip.clear();

                        Indx_h.clear();
                        Indx_Azm.clear();

            } //End of if (ValCheck.empty==1)

            CntlVal.clear();
            ValCheck.clear();

      } //End of for (i=0; i<n; i++)

      vector <int> FreqDat;
      outfile7<<"Pattern Number:"<<" "<<"Frequency:"<<" "<<"Pattern
Length:"
<<"Decimals:"<<endl;
      //Get pattern frequency//
      for (int ct=0; ct<Decimals.size(); ct++) {
            if (ct!=0){
                  if (findE(Decimals.at(ct),FreqDat)==0) {
                        int fre= GetFreq(Decimals.at(ct),Decimals) ;
                        FreqDat.push_back(Decimals.at(ct));
                        outfile7<<ct<<" "
                              <<fre<<" "
                              <<PatLeng.at(ct)<<" "
                              <<Decimals.at(ct)<<endl;}
            }
            else {
                  FreqDat.push_back(Decimals.at(ct));
                  int fre= GetFreq(Decimals.at(ct),Decimals) ;
                  outfile7<<ct<<" "
                              <<fre<<" "
                              <<PatLeng.at(ct)<<" "
                              <<Decimals.at(ct)<<endl;
            }
      }
      outfile7<<"end"<<endl;
      outfile4<<"end"<<endl;

      outfile.close();
      outfile2.close();
      outfile4.close();
      outfile5.close();
      outfile6.close();
      outfile7.close();
      outfile8.close();

      outfile3<<"Number of connections="<<count<<endl;

      outfile3.close();
```

```
      return 0;
} //end of main
```

Appendix B: Pattern Simulation Algorithm

B.1 OVERVIEW

The pattern simulation algorithm is presented in Chapter 4. The algorithm is developed in C++ using Microsoft Visual Studio 2008.The following input files are required:

i) Input file including the number of templates to be used, template descriptions, pattern histograms and pattern files. First line is the number of templates to be used in the pattern simulation. Then, template description files are given (a sample template file is shown in Figure A.2). Second group of files includes the pattern histograms as shown previously in Figure A.4. Third group is template nodes files (Figure A.5) and the last input file is highest frequency pattern to be applied at the conditioning data locations (Figure B.2). This pattern is selected among the templates used in the simulation.

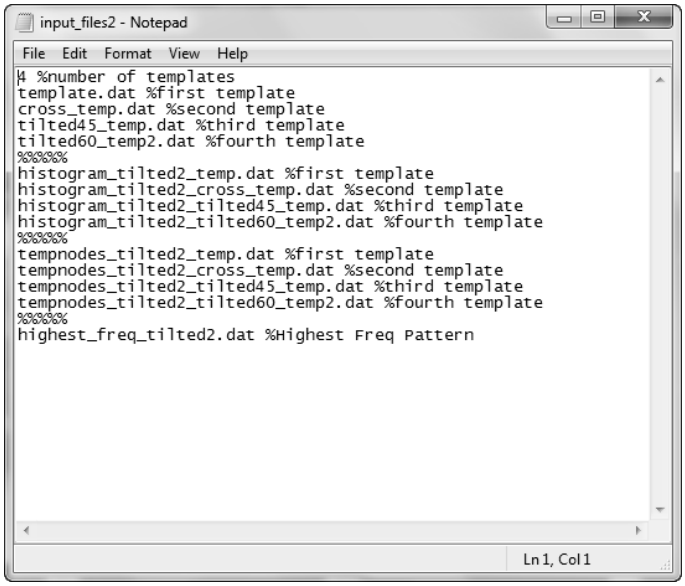


Figure B.1- Sample Pattern Simulation Input File. This example uses 4 different templates and corresponding pattern histograms.

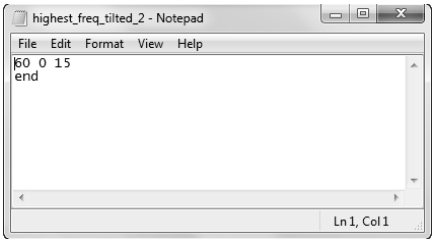


Figure B.2- Sample "highest frequency pattern" File. The pattern is defined by nodes by following the same notation as the template description given in Figure A.2.

ii) File with conditioning data locations. This file includes the coordinates of the conditioning data points.

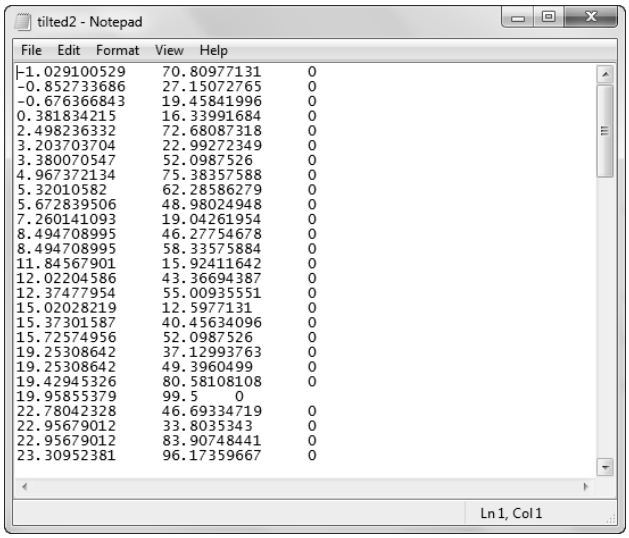


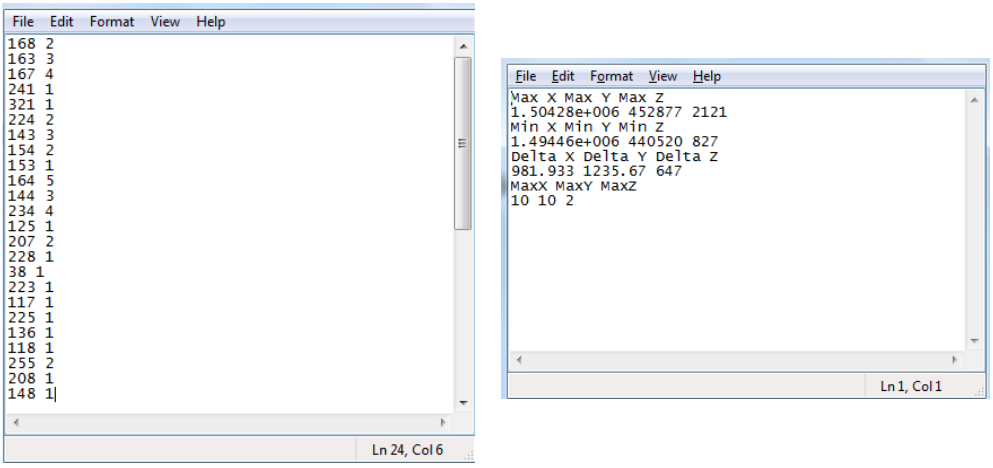
Figure B.3- Sample Conditioning Data File. The columns are x, y and z coordinates.

The following files are optional:

i) Quadrant frequency file. The first column includes the quadrant index number whereas the second one gives the number of points present in that particular quadrant (Figure B.4a).

ii) Quadrant properties file (Figure B.4b). This file includes the quadrant properties; boundary information of the given data set, quadrant size

(Delta X, Y and Z) and the number of quadrants in each direction (MaxX, MaxY and MaxZ).

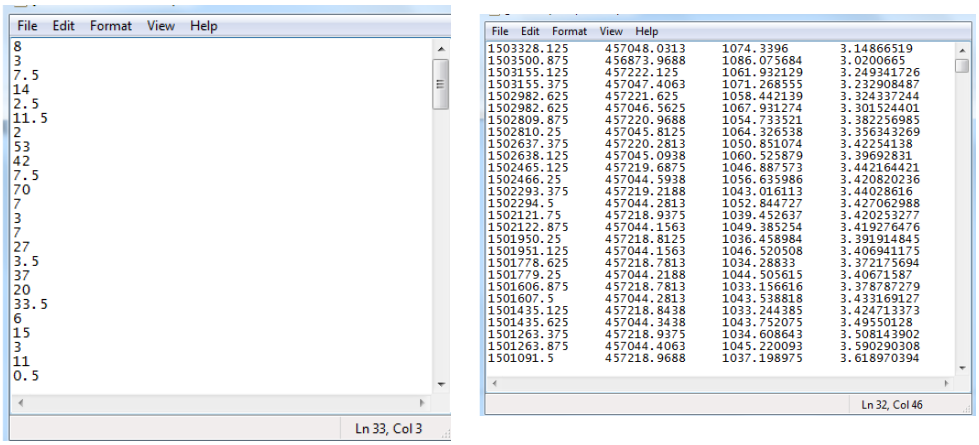


a-Quadrant Frequency File b- Quadrant Properties File

Figure B.4- Quadrant Information Files.

iii) Cave zone thickness values at the conditioning data locations. This file includes the cave zone thickness values only (Figure B.5a). The number of values should be equal to the number of conditioning data locations.

iv) Host formation dip map (Figure B.5b). The dip map is given by x, y, z coordinates and formation dip value at that particular location.



a- Cave Zone Thickness File b- Host Formation Dip Map

Figure B.5- Optional Files for Pattern Simulation

Once the input files and input parameters are provided, the pattern simulation algorithm is executed. The number of nodes in the highest frequency pattern file is provided through the command window (Figure B.6). This number is equal the number of nodes given in the "highest frequency pattern" file (Figure B.2).

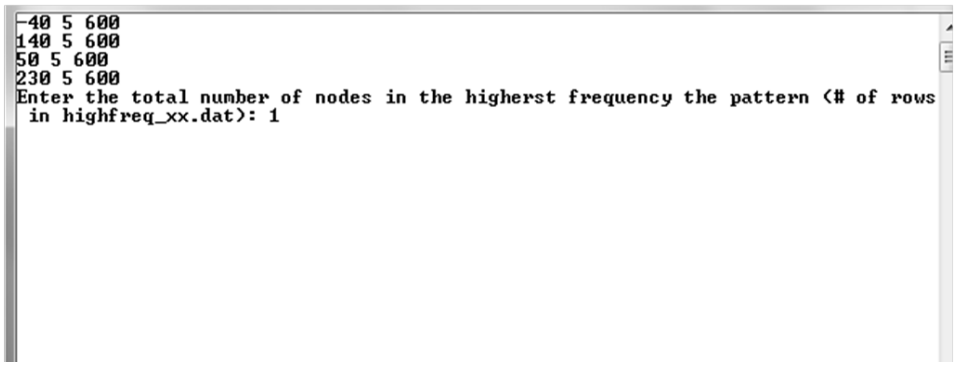


Figure B.6- Command Window Snap-shot for the Pattern Simulation

Output files of the pattern simulation program are conditioning data locations, simulated node coordinates (for both cave and non-cave nodes are stored), simulated cave network connections in line stick and Petrel general line formats, file storing simulated node connections by indices, simulated cave zone thicknesses (one file with thickness

only, one file with thickness and estimation variances for data transformation if needed), cave zone thickness map including discretized nodes along the simulated passages, file including the simulated passage azimuth angle and file storing the indices of the patterns retrieved from the histogram during the simulation.

B.2 HEADER AND FUNCTION FILES

B.2.1 Header Files for Reading Input Data

```
//"read_patterns.h" file
//Header file for reading patterns (temp_node_files)
#ifndef read_patterns
#define read_patterns

#include <iostream>
#include <sstream>
#include <string>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <iomanip>
#include <cmath>

using namespace std;

class My_TNode {
      public:
            vector<vector<int>> Nodes;
};
//Reads Entire Set of Patterns (temp_node_file) Given in Input File
vector <My_TNode> Read_Pat (ifstream& myfile) {

      string my_line;
      double num,val;
      int a;

      getline(myfile,my_line, '%'); //Read Number of Templates
      num= atoi(my_line.c_str()); //num:Number of templates

      vector<My_TNode>  TempNodes  (num);  //Stores  Histogram  (Pattern
Number, Frequency, Pattern Length, Decimals)

      getline(myfile,my_line, '\n'); //Read comment
      for (int ix=0; ix<2*num+1; ix++) {
            getline(myfile,my_line, '%'); //Read file name
            getline(myfile,my_line, '\n'); //Read comment
      }

      int numct=2*num; //Counter for num
```

```
      while(numct!=3*num) {

            getline(myfile,my_line, '\n'); //Read comment
            getline(myfile,my_line, '%'); //Read file name

            if(!my_line.empty()) { //If the line is not blank
                  ifstream file1(my_line.c_str());

                        if(fopen(my_line.c_str(),"r")==0) {
                        cout << my_line + "was not found.Please check
                                                the directory"<<endl;
                        file1.clear(ios::failbit);
                        }

                        else { //Start Reading and Storing the Template
Data

                              //Start Storing Template Node files
                                    string file_line;
                                    int ctr=0;
                                    while (!file1.eof())  {

                                    getline(file1,file_line,      '\n');
//Reads first line

                                    if (file_line=="end") { break;}
                                    char *lines;
                                    lines=new char [file_line.size()];
                                    strcpy(lines, file_line.c_str());
                                    char * TChar;
                                    TChar = strtok(lines," ");
                                    TempNodes.at(numct-
                              2*num).Nodes.push_back(vector<int> ());

                                          while (TChar!=0){
                                          TempNodes.at(numct-
2*num).Nodes[ctr].push_back(atof(TChar));
                                          TChar = strtok(NULL," ");
                                          }
                                          ctr=ctr+1;
                                    }

                        numct=numct+1;
                        file1.close();
                        } //End of else "//Start Reading and Storing
the Data"

            } //End of "If the line is not blank"
      } //End of while(!myfile.eof())
      return TempNodes;
}

//=====================================================================//
```

```
#endif //read_patterns

//"read_templates.h" file
//Header file for reading template files

#ifndef Read_Templates
#define Read_Templates

#include <iostream>
#include <sstream>
#include <string>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <iomanip>
#include <cmath>

using namespace std;

class My_Temp {

public:
      vector<double> AzmAng;
      vector<double> DipAng;
      vector<double> Dist;

};
//Reads Entire Set of Templates Given in Input File
vector <My_Temp> Read_Temp  (ifstream& myfile) {

string my_line;
double num,val;
int a;

getline(myfile,my_line, '%'); //Read Number of Templates
num= atoi(my_line.c_str()); //num:Number of templates
vector<My_Temp>  Templates  (num);  //Stores  Templates'  information
(Azm,Dip and Lag)

int numct=0; //Counter for num

while(numct!=num) {

getline(myfile,my_line, '\n'); //Read comment
getline(myfile,my_line, '%'); //Read file name

if(!my_line.empty()) { //If the line is not blank
      ifstream file1(my_line.c_str());

            if(fopen(my_line.c_str(),"r")==0) {
                        cout << my_line + "was not found.Please check
the directory"<<endl;
                        file1.clear(ios::failbit);
```

```
            }

            else { //Start Reading and Storing the Template Data

                  //Start Storing Templates
                  if (numct<num) {
                        file1>>a; //Reads template size

                        Templates.at(numct).AzmAng.resize(a);
                        Templates.at(numct).DipAng.resize(a);
                        Templates.at(numct).Dist.resize(a);

                              for (int i=0; i<a; i++) {

                                    file1>>val;

      Templates.at(numct).AzmAng.at(i)=val;
                                    cout<<val<<" ";

                                    file1>>val;

      Templates.at(numct).DipAng.at(i)=val;
                                    cout<<val<<" ";

                                    file1>>val;
                                    Templates.at(numct).Dist.at(i)=val;
                                    cout<<val<<endl;
                              }

                  } //End of Template Reading

            numct=numct+1;
            file1.close();

            } //End of else "//Start Reading and Storing the Data"
} //End of "If the line is not blank"

} //End of while(!myfile.eof())

return Templates;
}
//=====================================================================//
#endif //Read_Templates


//"read_histograms.h" file
//Header file for reading histogram files
#ifndef read_histograms
#define read_histograms

#include <iostream>
#include <sstream>
#include <string>
#include <fstream>
```

```
#include <stdio.h>
#include <stdlib.h>
#include <iomanip>
#include <cmath>

using namespace std;

class My_Hist {
public:
      vector <double> PatNum;
      vector <double> Freq;
      vector <double> PatLen;
      vector <double> Decs;
};

//Reads Entire Set of Histograms Given in Input File
vector <My_Hist> Read_Hist (ifstream& inputfilen) {

string my_line;
double num,val;
int a;

getline(inputfilen,my_line, '%'); //Read Number of Templates
num= atoi(my_line.c_str()); //num:Number of templates

vector<My_Hist> Histograms (num); //Stores Histogram (Pattern Number,
Frequency, Pattern Length, Decimals)

getline(inputfilen,my_line, '\n'); //Read comment
for (int ix=0; ix<num; ix++) {
      getline(inputfilen,my_line, '%'); //Read file name
      getline(inputfilen,my_line, '\n'); //Read comment
}

int numct=num; //Counter for num

while(numct!=2*num) {
      getline(inputfilen,my_line, '\n'); //Read comment
      getline(inputfilen,my_line, '%'); //Read file name

      if(!my_line.empty()) { //If the line is not blank
            ifstream file1(my_line.c_str());
                  if(fopen(my_line.c_str(),"r")==0) {
                              cout << my_line + "was not found.Please
check the directory"<<endl;
                              file1.clear(ios::failbit);
                  }

                  else { //Start Reading and Storing the Template Data

                        //Start Storing Histograms
                        if (numct<2*num && numct>=num) {
                              string file_line;
```

```
                              getline(file1,file_line,   '\n');   //Reads
first line
                              while (!file1.eof())  {

                                    getline(file1,file_line, '\n');

                                    if (file_line=="end") { break;}

                                    vector <double> klm;
                                    char *lines;
                                    lines=new char [file_line.size()];
                                    strcpy(lines, file_line.c_str());
                                    char * TChar;
                                    TChar = strtok(lines," ");
                                    klm.push_back(atof(TChar));

                                    Histograms.at(numct-
num).PatNum.push_back(klm.at(0));
                                    for (int i=0; i<3; i++){
                                          TChar = strtok(NULL," ");
                                          klm.push_back(atof(TChar));
                                                }
                                                Histograms.at(numct-
num).Freq.push_back(klm.at(1));
                                                Histograms.at(numct-
num).PatLen.push_back(klm.at(2));
                                                Histograms.at(numct-
num).Decs.push_back(klm.at(3));
                                    }
                        } //End of Histogram Storage
                  numct=numct+1;
                  file1.close();
                  } //End of else "//Start Reading and Storing the
Data"
      } //End of "If the line is not blank"

} //End of while(!inputfilen.eof())

return Histograms;
}

//=====================================================================//
#endif // read_histograms
```

B.2.2 Header and Function Files for Pattern Simulation

```
//"functions.h" header file
//This file includes functions used in pattern simulation algorithm.

//______globals.h and random number generator from________
// GsTL: the Geostatistics Template Library
// Author: Nicolas Remy
```

```
// Copyright (c) 2000 The Board of Trustees of the Leland Stanford
Junior University
// All rights reserved.

#ifndef functions
#define functions

#include "read_histograms.h"
#include <time.h>
#include <ctime>
#include <windows.h>
#include "globals.h"

using namespace std;

class RetInd {
  public:
      vector <int> Ta;
      vector <int> Tb;
};

class Temp {
  public:
      double AzmAng;
      double DipAng;
      double Dist;
};

//***Maximum and Minimum X,Y,Z coordinates of Cave Data
vector  <double>  bounds(double  **Array,int  Select,  double  numD)  {
//Array = Data
#include <vector>
      vector <double> Vals (3);   //[Xmin, Ymin, Zmin] //[Xmax, Ymax,
Zmax]

      Vals.at(0)=Array[0][0];
      Vals.at(1)=Array[0][1];
      Vals.at(2)=Array[0][2];
      //Select=1 for minimum, Select=2 for maximum
      switch (Select) {
            case 1: //Determine Minimum
                  for (int t=1; t<numD; t++) {
                        //Minimum X Coordinate
                        if (Array[t][0]<Vals.at(0)) {
                        Vals.at(0)=Array[t][0];
                        }
                        //Minimum Y Coordinate
                        if (Array[t][1]<Vals.at(1)) {
                        Vals.at(1)=Array[t][1];
                        }
                        //Minimum Z Coordinate
                        if (Array[t][2]<Vals.at(2)) {
                        Vals.at(2)=Array[t][2];
                        }
```

```
                  }
                  break;

            case 2: //Determine Maximum
                  for (int t=1; t<numD; t++) {
                        //Minimum X Coordinate
                        if (Array[t][0]>Vals.at(0)) {
                        Vals.at(0)=Array[t][0];
                        }
                        //Minimum Y Coordinate
                        if (Array[t][1]>Vals.at(1)) {
                        Vals.at(1)=Array[t][1];
                        }
                        //Minimum Z Coordinate
                        if (Array[t][2]>Vals.at(2)) {
                        Vals.at(2)=Array[t][2];
                        }
                  }
                  break;
      }
      return Vals;
}

//****Boundary Control**//
int  BndContl(vector<double>Node,vector<double>MinB,vector<double>  MaxB)
{
      int BndCond=0; //BndCond=0: Outside the boundary
      if (Node[0]>=MinB.at(0) && Node[0]<=MaxB.at(0)) {
            if (Node[1]>=MinB.at(1) && Node[1]<=MaxB.at(1)) {
                  if (Node[2]>=MinB.at(2) && Node[2]<=MaxB.at(2)) {
                        BndCond=1;}
                  else {BndCond=0;}
            }
            else {BndCond=0;}
      }
      else {BndCond=0;}
      //cout<<"Bound Cont="<<BndCond<<endl;
      return BndCond;
}

//**Converts degrees to radians
double deg2rad(double Deg) {
double pi_a= 3.14159;
double Rad=(Deg*pi_a)/180;
return Rad;}

//**Calculation of Difference**//
double Delta(double Loc1,double Loc2)
{ double dif;
dif=fabs(Loc1-Loc2);
return dif;}

//**Calculation of Lag Distance**//
```

```
double Cal_h(double Loc1X,double Loc1Y,double Loc1Z,double Loc2X,double
Loc2Y,double Loc2Z){
      double CalH;
      double DX=Delta(Loc1X,Loc2X);
      double DY=Delta(Loc1Y,Loc2Y);
      double DZ=Delta(Loc1Z,Loc2Z);

      CalH=sqrt(pow(DX,2)+pow(DY,2)+pow(DZ,2));
      return CalH;}

//****Calculation of Distance (3D)**//
double Cal_d(double Dis1,double Dis2)
{ double Dis3;
      Dis3=sqrt(pow(Dis1,2)+pow(Dis2,2));
      return Dis3;}

//***Calculates Azimuth Angle
double Cal_Azm(double Loc1X,double Loc1Y,double Loc1Z,double
Loc2X,double Loc2Y,double Loc2Z){
#include <cmath>
double Azm_Ang=0;
double Dist=Cal_d(fabs(Loc1X-Loc2X),fabs(Loc1Z-Loc2Z));
double LagD=Cal_h(Loc1X,Loc1Y,Loc1Z,Loc2X,Loc2Y,Loc2Z);
double pi_a= 3.14159;

if (Loc1Y!=Loc2Y){
      if(Loc2Y>Loc1Y) {
            if (Loc2X>Loc1X) {
                  Azm_Ang=atan(Dist/(Loc2Y-Loc1Y)); }
            if (Loc2X<Loc1X) {
                  Azm_Ang=2*pi_a-fabs(atan(Dist/(Loc2Y-Loc1Y)));  }
            if (Loc2X==Loc1X || fabs(Loc2X-Loc1X)<=0.0001) {
            Azm_Ang=0;  }
      }
      else {
            if (Loc2X>Loc1X) {
                  Azm_Ang=pi_a-fabs(atan(Dist/(Loc2Y-Loc1Y))); }
            if (Loc2X<Loc1X) {
                  Azm_Ang=pi_a+fabs(atan(Dist/(Loc2Y-Loc1Y))); }

            if (Loc2X==Loc1X || fabs(Loc2X-Loc1X)<=0.0001) {
            Azm_Ang=pi_a; }
      }
}
else {
      if(Loc2X==Loc1X || fabs(Loc2X-Loc1X)<=0.0001) {
            if(Loc2Z>Loc1Z) {
            Azm_Ang=pi_a/2;}
            if(Loc2Z<Loc1Z) {
            Azm_Ang=3*pi_a/2;}
      }
      else {
            if(Loc2X>Loc1X) {
            Azm_Ang=pi_a/2; }
```

```
            if(Loc2X<Loc1X) {
            Azm_Ang=3*pi_a/2;}
      }
}
      return Azm_Ang;
}

//Calculates Dip Angle
double    Cal_Dip(double    Loc1X,double    Loc1Y,double    Loc1Z,double
Loc2X,double Loc2Y,double Loc2Z){
#include <cmath>
double Dip_Ang;
double Dist=Cal_d(fabs(Loc1X-Loc2X),fabs(Loc1Z-Loc2Z));
double DelZ=Delta(Loc1Z,Loc2Z);
double pi_a= 3.14159;

      if (Loc1Z!=Loc2Z) {
            Dip_Ang=atan((Loc2Z-Loc1Z)/(Loc2X-Loc1X));}//Dip angle
      else { Dip_Ang=0; }
return Dip_Ang;
}

//**Find whether an Element is in a Vector or not : For Int Values**//
int findE(int vec1,vector <int> vec2) {
      #include <vector>
      //cond=1 : element is in the vector. cond=0 : element is not in
the vector
      int cond=0;
      int cts=0;
      for (int kl=0; kl<vec2.size(); kl++) {
                  if (vec1==vec2.at(kl)) {
                        cts++;
                        if (cts>0) {
                        cond=1;
                        break; }
                  }
            }
return cond;
}

//**Find whether an Element is in a Vector or not : For Double
Values**//
int findED(double vec1,vector <double> vec2) {
      #include <vector>
      //cond=1 : element is in the vector. cond=0 : element is not in
the vector
      int cond=0;
      int cts=0;
      for (int kl=0; kl<vec2.size(); kl++) {
                  if (vec1==vec2.at(kl)) {
                        cts++;
                        if (cts>0) {
                        cond=1;
                        break; }
```

```
                  }
            }
return cond;
}

//***Find Indices in Array for given Condition**//
int findI(int vec1,vector <int> vec2) {
      int cond;
      int m=0;
      for (int kl=0; kl<vec2.size(); kl++) {
                  if (vec1==vec2.at(kl)) {
                  cond=kl;}
            }
return cond;
}

//***Find Indices in Array for CondTemp ***//
vector <int> find_temp(double dist,double dist_tol,double rad,double
rad_tol,double rdip,double dip_tol,vector <double> vec_dist, vector
<double> vec_ang,vector <double> vec_dip) {
      #include <vector>
      double pi_a= 3.14159;
      vector <int> cond;
      double rad_low,azmval;

      for (int kl=0; kl<vec_dist.size(); kl++) {
            azmval=vec_ang.at(kl);
            if (dist-dist_tol<=vec_dist.at(kl) &&
            vec_dist.at(kl)<=dist+dist_tol && rad-rad_tol/2<=azmval &&
            azmval<=rad+rad_tol/2 && rdip-dip_tol/2<=vec_dip.at(kl) &&
            vec_dip.at(kl)<=rdip+dip_tol/2) {
                        cond.push_back(kl);}
            }
      return cond;
}

//***Search Angle Data Radially ***//
RetInd search_temp(vector< vector<double>> CTemp,int sz,vector<double>
LagAr,vector<double> AzmAr, vector <double> DipAr,double DistTol,
double AzmTol, double DipTol) {     //sz:Template size
#include <vector>
      vector <int>Indx(0);
      vector<int>Tempo;
      vector <int> TempInd;
      double pi_a= 3.14159;
      double DistF=DistTol;

      AzmTol=deg2rad(AzmTol);
      DipTol=deg2rad(DipTol);

      int tl,rnd;
      for (tl=0; tl<sz; tl++) { //Loop over CTemp
            if (CTemp[tl][0]<0) {
                  CTemp[tl][0]=2*pi_a-fabs(CTemp[tl][0]);}
```

```
            DistTol=DistF*CTemp[tl][2];

      Tempo=find_temp(CTemp[tl][2],DistTol,CTemp[tl][0],AzmTol,CTemp[tl
][1],DipTol,LagAr,AzmAr,DipAr);

            if (Tempo.empty()!=1) {
                  rnd=0;
                  if (Tempo.size()>1) {
                        rnd= (Tempo.size())*rand_generator_();}

                  if (findE(Tempo.at(rnd),Indx)==0) {
                        TempInd.push_back(tl);
                        Indx.push_back(Tempo.at(rnd));}
            Tempo.clear();
            }
            DistTol=0;
      }//End of Loop over CTemp: for (tl=0; tl<sz; tl++)

      RetInd ArrRet;
      ArrRet.Ta=Indx;
      ArrRet.Tb=TempInd;
      return ArrRet;
}

//Dot Product of Two Vectors: Results similarity between V1 and V2
int DotProd(vector<int> V1, vector<int> V2) {
      int tl;
      int DProd;
            //Initial Check: Vector sizes should be equal
            if (V1.size()!=V2.size()) {
            cout<<"Vector sizes should be equal."<<endl;}
            else { DProd=0;
                  for (tl=0; tl<V1.size(); tl++) {
                        //DProd+=V1.at(tl)*V2.at(tl);
                        if (V1.at(tl)==V2.at(tl)) {
                              DProd+=1;}
                  }
            }
      return DProd;
}

//Find index of the maximum element
vector<int> findmax(vector <int> Vec) {
      int kt;
      vector <int> km;
      int Valmax=Vec.at(0);//=max_element(Vec.begin(),Vec.end());
      km.push_back(0);

      if (Vec.size()!=1) {
            for (kt=1; kt<Vec.size();kt++) {
                  if (Valmax==Vec.at(kt)) {
                        km.push_back(kt);}
```

```
                  if (Valmax<Vec.at(kt)) {
                        km.clear();
                        Valmax=Vec.at(kt);
                        km.push_back(kt);}
            }
      }
      else {
            km.push_back(0);}
return km;
}

//Find indeces of elements for the given condition
vector<int> findany(vector <double> Vec, double MyVal, int MyCond) {
      //MyCond=0 Equal to; MyCond=1; Smaller than; MyCond=2; Greater
than
      int kt;
      vector <int> km;

      for (kt=0; kt<Vec.size();kt++) {
            if (MyCond==0) {
                  if (Vec.at(kt)==MyVal) {
                        km.push_back(kt);}
            }

            if (MyCond==1) {
                  if (Vec.at(kt)<=MyVal) {
                        km.push_back(kt);}
            }

            if (MyCond==2) {
                  if (Vec.at(kt)>=MyVal) {
                        km.push_back(kt);}
            }
      }
return km;}
//Find index of the maximum element
int findminD(vector <double> Vec) {
      int kt;
      int km;
      vector <int> indkm;

      double Valmin=Vec.at(0);
      indkm.push_back(0);

      if (Vec.size()!=1) {
            for (kt=1; kt<Vec.size();kt++) {

                  if (Valmin==Vec.at(kt)) {
                        indkm.push_back(kt);}

                  if (Valmin>Vec.at(kt)) {
                        indkm.clear();
                        Valmin=Vec.at(kt);
                        indkm.push_back(kt);}
```

```
            }
      }
//!!!!!!Modification for SelectPat Function!!!!!
      if (indkm.size()>1) {
            int rind=(indkm.size())*rand_generator_();
            km=indkm.at(rind);}
      else {
            km=indkm.back();}
return km;
}

//Find index of the maximum element
int findmaxD2(vector <double> Vec) {
      int kt;
      int km;
      vector <int> indkm;
      double Valmax=Vec.at(0);
      indkm.push_back(0);

      if (Vec.size()!=1) {
            for (kt=1; kt<Vec.size();kt++) {
                  if (Valmax==Vec.at(kt)) {
                        indkm.push_back(kt);}

                  if (Valmax<Vec.at(kt)) {
                        indkm.clear();
                        Valmax=Vec.at(kt);
                        indkm.push_back(kt);}
            }
      }
//!!!!!!Modification for SelectPat Function!!!!!
      if (indkm.size()>1) {
            int rind= (indkm.size())*rand_generator_();
            km=indkm.at(rind);}
      else {
            km=indkm.back();}
return km;
}

//Find index of the maximum element
vector <int> findmaxD(vector <double> Vec) {
      int kt;
      int km;
      vector <int> indkm;
      double Valmax=Vec.at(0);
      indkm.push_back(0);

      if (Vec.size()!=1) {
            for (kt=1; kt<Vec.size();kt++) {

                  if (Valmax==Vec.at(kt)) {
                        indkm.push_back(kt);}

                  if (Valmax<Vec.at(kt)) {
```

```
                        indkm.clear();
                        Valmax=Vec.at(kt);
                        indkm.push_back(kt);}
            }
      }
return indkm;
}

//Select Pattern with Max. Similarity and Highest Frequency
vector <int> SelectPat (vector<int> MaxIndV,vector<vector<int>> Vals,
vector<My_Hist> RHist, int A) {  //PatVals, ReadHist, NTemp
      int km;
      vector <double> Freq;

      for (km=0; km<MaxIndV.size(); km++) {
            int TNum=Vals.at(MaxIndV.at(km)).at(0);  //Template Number
            int PatNum=Vals.at(MaxIndV.at(km)).at(1); //Pattern Index
            double    PatFreq=RHist.at(TNum).Freq.at(PatNum);    //Store
Pattern Frequency
            Freq.push_back(PatFreq);}

      vector <int> indkm;
      indkm=findmaxD(Freq);
      vector <int> MyPat;

      if (indkm.size()>1) {
            int      rind=      (indkm.size())*rand_generator_();//rand()
%indkm.size();
            km=Vals.at(MaxIndV.at(rind)).at(0); //indkm.at(rind);
            MyPat.push_back(km); //Return Template Number
            MyPat.push_back(Vals.at(MaxIndV.at(rind)).at(1));}  //Return
Pattern Index
      else {
            km=Vals.at(MaxIndV.at(indkm.back())).at(0); //indkm.back()
            MyPat.push_back(km); //Return Template Number
            MyPat.push_back(Vals.at(MaxIndV.at(indkm.back())).at(1));
}//Return Pattern Index

return MyPat;
}

//***Calculation of New Node Coordinates for Given Parameters of Lag
Dist.,Azimuth and Dip Angles***//
double   *BackCal(double   AzmAng,double   DipAng,double   SepDist,double
X1,double Y1,double Z1)
{     double Y2,Dist1,X2,Z2;
      X2=X1;
      Y2=Y1;
      Z2=Z1;

      Dist1=SepDist*sin(deg2rad(AzmAng));
      X2=X1+Dist1*cos(deg2rad(DipAng));
      Y2=Y1+SepDist*cos(deg2rad(AzmAng));
      Z2=Z1+Dist1*sin(deg2rad(DipAng));
```

```
      double *Coord2=new double[3];
      Coord2[0]=X2;
      Coord2[1]=Y2;
      Coord2[2]=Z2;
      return Coord2;
}

//***Calculation of New Node Coordinates for Given Parameters of Lag
Dist.,Azimuth and Dip Angles***//
double *BackCal2(double AzmAng,double DipAng,double SepDist,double
X1,double Y1,double Z1,double DistTol, double AzmTol, double DipTol)
{     double Y2,Dist1,X2,Z2;
            X2=X1;
            Y2=Y1;
            Z2=Z1;

      DistTol=DistTol*SepDist;  //distance tolerance is given as a
percentage.

      int Range;
      Range=(DistTol);
      SepDist=Range*rand_generator_()+((SepDist-DistTol/2));

      Range=(AzmTol);
      AzmAng=Range*rand_generator_()+((AzmAng-AzmTol/2));

      Range=(DipTol);
      DipAng=Range*rand_generator_()+((DipAng-DipTol/2));

      Dist1=SepDist*sin(deg2rad(AzmAng));
      X2=X1+Dist1*cos(deg2rad(DipAng));
      Y2=Y1+SepDist*cos(deg2rad(AzmAng));

      if (Z1!=0) {
            Z2=Z1+Dist1*sin(deg2rad(DipAng)); }

      double *Coord2=new double[4];
      Coord2[0]=X2;
      Coord2[1]=Y2;
      Coord2[2]=Z2;
      Coord2[3]=AzmAng;
      return Coord2;
}

//***Calculation of New Node Coordinates for Conditioning Data
Locations***//
double *PatBackCal(int rind,int jv, double AzmAng,double DipAng,double
SepDist,double    X1,double    Y1,double    Z1,double    AzmAng2,double
DipAng2,double SepDist2)
{
      double Y2,Dist1,X2,Z2;
      double *Coord2=new double[6];
            X2=X1;
```

```
            Y2=Y1;
            Z2=Z1;

            //Regular Back Calculation.CondData is at the center
            if (rind!=jv) {
                  Dist1=SepDist*sin(deg2rad(AzmAng));

                  double Cx,Cy,Cz,Cd;
                  //Calculate Center Node Location
                  Cd=SepDist2*sin(deg2rad(AzmAng2));

                  int dni=cos(deg2rad(DipAng2));

                  Cx=X1-Cd*cos(deg2rad(DipAng2));
                  Cy=Y1-SepDist2*(cos(deg2rad(AzmAng2)));
                  Cz=Z1-Cd*sin(deg2rad(DipAng2));

                  Coord2[0]=X1;
                  Coord2[1]=Y1;
                  Coord2[2]=Z1;

                  Coord2[3]=Cx;
                  Coord2[4]=Cy;
                  Coord2[5]=Cz;
            }

            //CondData  is  at  the  nodes.Center  Node  Calculation  is
Required.
            else {
                  double Cx,Cy,Cz,Cd;
                  //Calculate Center Node Location
                  Cd=SepDist2*sin(deg2rad(AzmAng2));

                  int dni=cos(deg2rad(DipAng2));

                  Cx=X1-Cd*cos(deg2rad(DipAng2));
                  Cy=Y1-SepDist2*(cos(deg2rad(AzmAng2)));
                  Cz=Z1-Cd*sin(deg2rad(DipAng2));

                  //Back Calculate by using the Center Node
                  Dist1=SepDist*sin(deg2rad(AzmAng));

                  X2=Cx+Dist1*cos(deg2rad(DipAng));
                  Y2=Cy+SepDist*(cos(deg2rad(AzmAng)));
                  Z2=Cz+Dist1*sin(deg2rad(DipAng));

                  Coord2[0]=X2;
                  Coord2[1]=Y2;
                  Coord2[2]=Z2;

                  Coord2[3]=Cx;
                  Coord2[4]=Cy;
                  Coord2[5]=Cz;
```

```
            }
      return Coord2;
}

//***Calculation of New Node Coordinates for Conditioning Data
Locations***//
double *PatBackCal2(int rind,int jv, double AzmAng,double DipAng,double
SepDist,double    X1,double    Y1,double    Z1,double    AzmAng2,double
DipAng2,double SepDist2,double DistTol, double AzmTol, double DipTol)
{
      double Y2,Dist1,X2,Z2,DRan1,DRan2,AzRan1,AzRan2,DipRan1,DipRan2;
      double *Coord2=new double[7];
      X2=X1;
      Y2=Y1;
      Z2=Z1;

            //Regular Back Calculation.CondData is at the center
            if (rind==jv) {
                  Dist1=SepDist*sin(deg2rad(AzmAng));
                  double Cx,Cy,Cz,Cd;

                  //Calculate Center Node Location
                  Cd=SepDist2*sin(deg2rad(AzmAng2));

                  int dni=cos(deg2rad(DipAng2));

                  Cx=X1-Cd*cos(deg2rad(DipAng2));
                  Cy=Y1-SepDist2*(cos(deg2rad(AzmAng2)));
                  Cz=Z1-Cd*sin(deg2rad(DipAng2));

                  Coord2[0]=X1;
                  Coord2[1]=Y1;
                  Coord2[2]=Z1;

                  Coord2[3]=Cx;
                  Coord2[4]=Cy;
                  Coord2[5]=Cz;

                  Coord2[6]=AzmAng2;
            }
            //CondData  is  at  the  nodes.Center  Node  Calculation  is
Required.
            else {
                  double Cx,Cy,Cz,Cd;
                  //Calculate Center Node Location
                  Cd=SepDist2*sin(deg2rad(AzmAng2));
                  int dni=cos(deg2rad(DipAng2));

                  Cx=X1-Cd*cos(deg2rad(DipAng2));
                  Cy=Y1-SepDist2*(cos(deg2rad(AzmAng2)));
                  Cz=Z1-Cd*sin(deg2rad(DipAng2));
                  DistTol=DistTol*SepDist;

                  //Back Calculate by using the Center Node
```

```
                  DRan1=SepDist-DistTol;
                  DRan2=SepDist+DistTol;

                  double Range;
                  Range=(DistTol);
                  SepDist=Range*rand_generator_()+((SepDist-
DistTol/2));

                  Range=(AzmTol);
                  AzmAng=Range*rand_generator_()+((AzmAng-AzmTol/2));

                  Range=(DipTol);
                  DipAng=Range*rand_generator_()+((DipAng-DipTol/2));

                  Dist1=SepDist*sin(deg2rad(AzmAng));

                  X2=Cx+Dist1*cos(deg2rad(DipAng));
                  Y2=Cy+SepDist*(cos(deg2rad(AzmAng)));
                  Z2=Cz+Dist1*sin(deg2rad(DipAng));

                  Coord2[0]=X2;
                  Coord2[1]=Y2;
                  Coord2[2]=Z2;

                  Coord2[3]=Cx;
                  Coord2[4]=Cy;
                  Coord2[5]=Cz;
                  Coord2[6]=AzmAng;
            }
      return Coord2;
}

double sumall(vector <double> Vec)
{ double Res=0;
      for (int is=0; is<Vec.size(); is++) {
            Res=Res+Vec.at(is);     }
      return Res;
}

//Extracting Frequencies by using Decimal Data***//
int GetFreq(int Data,vector <int>DataVec) {
      int F=0;
      for (int s=0; s<DataVec.size(); s++) {
            if (Data==DataVec.at(s)) {
                  F++;}
      }
return F;
}

//Convert integer to string. From cplusplus.com/forum
string convertInt(int number){
   stringstream ss;//create a stringstream
   ss << number;//add number to the stream
   return ss.str();}//return a string with the contents of the stream
```

```
template <class Data>
vector <Data>CalLog(vector <Data> Vec1) {
      vector <Data> VecRet (Vec1.size());
            for (int tl=0; tl<Vec1.size(); tl++) {
                  if (Vec1.at(tl)!=0) {
                        VecRet.at(tl)=log(Vec1.at(tl));}
            }
      return VecRet;
}

double find_temp_dip(double XL, double YL, double ZL, vector <vector
<double>> MM) {
      vector <double> DSS;

      for (int kklm=0; kklm<MM.size(); kklm++) {

      DSS.push_back(Cal_h(XL,YL,ZL,MM.at(kklm).at(0),MM.at(kklm).at(1),
MM.at(kklm).at(2)));}

      int myidx=findminD(DSS);
      double DV=MM.at(myidx).at(3);
      return DV;
}

//Reads dip map values if given.
vector <vector <double>> read_dip_map(istream &MapFile) {
//dip values should be given in degrees. deg2rad conversion is done
inside the algorithm.
vector <vector <double> > DipM;
double mya;

while (!MapFile.eof()){
      DipM.push_back(vector <double> (4));

      for (int ikl=0; ikl<4; ikl++) {
      MapFile>>mya;
      DipM.back().at(ikl)=mya;

      if (ikl==3) {
            DipM.back().at(ikl)=deg2rad(mya);}
      }
}
return DipM;
}

//=====================================================================//
#endif // functions


//"histogram_dist.h" file
//Calculates deviation from a refence histogram.
//This file is used for dynamically checking for histogram deviation.
```

```
#ifndef histogram_dist
#define histogram_dist

#include <iostream>
#include <sstream>
#include <string>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <iomanip>
#include <cmath>
#include "functions.h"
#include "read_histograms.h"

using namespace std;

class FreqVal {
  public:
      vector <int> DecVal;
      vector <double> Freq;
};


class UpdateRes {
  public:
      vector<double> X;
      vector<double> Y;
      vector<double> Z;
      vector <int> CNode;
      vector<int> FNode;
      vector<vector <int>> UpCond;
};

double Distance (vector <double> XVal,vector <double> YVal, vector
<double> ZVal, vector< vector <double> > Temp,vector<int> FVal, vector
<My_Hist> OrigHist, int HistInd,double DistTol, double AzmTol, double
DipTol )  {
      vector<double> Lag;
      vector<double> Azm; //Calculated Azimuth angle
      vector<double> Dip; //Calculated Dip angle
      vector<int> IndxJ; //J indices
      vector<int> Indx_Azm_Temp;
      vector<int> Indx_h_Temp;

      double HistD=0;

      vector<vector <int> > UpdateCon; //(Connec.size());

      //=====================================================//
      int j,count, ncon; //count:number of connections
      count=0;
      ncon=0;
      double azmr;
```

```
      RetInd Arr;

      //=============================================//
      double fnum=0;
      vector <int> Decimals; //stores decimals of the patterns
      vector <int> PatLeng; //stores pattern length
      vector  <  vector  <int>  >  NodeCon;  //store  the  determined
connections
       //=============================================//
      vector <UpdateRes> RetVals (1);
      int iv;
      /**=====================Analysis Starts=================**/

      for (iv=0; iv<XVal.size(); iv++) {
            if (FVal.at(iv)==1) {

                  for (j=0; j<XVal.size(); j++) {
                        if (j!=iv && FVal.at(j)==1) {

      Lag.push_back(Cal_h(XVal[iv],YVal[iv],ZVal[iv],XVal[j],YVal[j],ZV
al[j]));
                        IndxJ.push_back(j);

      Azm.push_back(Cal_Azm(XVal[iv],YVal[iv],ZVal[iv],XVal[j],YVal[j],
ZVal[j]));
      Dip.push_back(Cal_Dip(XVal[iv],YVal[iv],ZVal[iv],XVal[j],YVal[j],
ZVal[j]));}
                  }
      Arr=search_temp(Temp,Temp.size(),Lag,Azm,Dip,DistTol,AzmTol,DipTo
l);

                  if (Arr.Ta.empty()==0) {
                        Indx_h_Temp=Arr.Ta;
                        NodeCon.push_back(vector <int> ());
                        NodeCon.at(NodeCon.size()-1).push_back(iv);

      //Converts Observed pattern into Decimal (to get frequency)//
      //Constructs the histogram//
                        for (int ct=0; ct<Arr.Tb.size(); ct++) {
                              NodeCon.at(NodeCon.size()-
1).push_back(IndxJ.at(Arr.Ta.at(ct)));

                              fnum=fnum+pow(double (2),Arr.Tb.at(ct));

                              if (Arr.Tb.size()-ct==1) {
                                    Decimals.push_back(fnum);
                                    PatLeng.push_back(Arr.Tb.size());}
                        }

                  }

                  fnum=0;

                  Lag.clear();
```

```
                  IndxJ.clear();
                  Azm.clear();
                  Dip.clear();

                  Indx_Azm_Temp.clear();
                  Indx_h_Temp.clear();}
      } //End of for (iv=0; iv<n; iv++)

      vector <FreqVal> FreqDat(1);

      //Get pattern frequency//
      for (int ct=0; ct<Decimals.size(); ct++) {

            if (ct!=0){
                  if (findE(Decimals.at(ct),FreqDat.at(0).DecVal)==0) {
                  FreqDat.at(0).DecVal.push_back(Decimals.at(ct));
                        int fre= GetFreq(Decimals.at(ct),Decimals) ;
                        FreqDat.at(0).Freq.push_back(fre);
                   }
            }
            else {
                  FreqDat.at(0).DecVal.push_back(Decimals.at(ct));
                  int fre= GetFreq(Decimals.at(ct),Decimals) ;
                  FreqDat.at(0).Freq.push_back(fre);
            }
      }

      double SumSim=sumall(FreqDat.at(0).Freq);
      double SumOrig=sumall(OrigHist.at(HistInd).Freq);

      for (int is=0; is<FreqDat.at(0).Freq.size(); is++) {
      FreqDat.at(0).Freq.at(is)=FreqDat.at(0).Freq.at(is)/SumSim;}

      for (int is=0; is<OrigHist.at(HistInd).Freq.size(); is++) {

      OrigHist.at(HistInd).Freq.at(is)=OrigHist.at(HistInd).Freq.at(is)
/SumOrig;}

      vector<int>::iterator it;
      int lks;
      double DisVal=0;
      //Calculate Euclidean Distance between the Original and Simulated
Histograms //
      for (int ct=0; ct<OrigHist.at(HistInd).Freq.size(); ct++) {
            it =
            search_n(FreqDat.at(0).DecVal.begin(),FreqDat.at(0).DecVal.
            end(),1,OrigHist.at(HistInd).Decs.at(ct));
            lks= int (it-FreqDat.at(0).DecVal.begin());

            if (lks!=FreqDat.at(0).DecVal.size()){
            DisVal=DisVal+pow(fabs(FreqDat.at(0).Freq.at(lks)-
OrigHist.at(HistInd).Freq.at(ct)),2);
            }
            else {
```

```
      DisVal=DisVal+pow(fabs(OrigHist.at(HistInd).Freq.at(ct)),2);}
      }

      HistD=pow(DisVal,0.5);

return HistD;

}
      //=============================================================//
#endif // dynamic_histogram

//Globals Header File modified from GsTL
#ifndef _globals_h
#define _globals_h

#include "random_number_generators.h"
Rand48_generator rand_generator_;
#endif _globals_h

//”random_number_generators.h” header file
/* GsTL: the Geostatistics Template Library
 *
 * Author: Nicolas Remy
 * Copyright (c) 2000 The Board of Trustees of the Leland Stanford
Junior University
 *
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
modification,
 * are permitted provided that the following conditions are met:
 *
 *1.Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
 *2.Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
 *3.The name of the author may not be used to endorse or promote
products derived from this software without specific prior written
permission.

 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN
NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.
 */
```

```
#ifndef __GSTL_RANDOM_NUMBER_GENERATORS_H__
#define __GSTL_RANDOM_NUMBER_GENERATORS_H__

#include <boost/random/mersenne_twister.hpp>

#include <cmath>
#include <stdlib.h>

struct Rand48_generator {
  typedef boost::mt11213b::result_type IntType;
  typedef double return_type;

  Rand48_generator() {}
  Rand48_generator( long int seed ) : gen( static_cast<IntType>(seed) )
{}
inline void seed( long int seed ) { gen.seed(
static_cast<IntType>(seed) ); }
inline double operator() () { return double( gen() ) / double(
gen.max() ); }

private:
      boost::mt11213b gen;

};

/*
struct Rand48_generator{

  typedef double return_type;
  Rand48_generator() {}
  Rand48_generator(long int seed) {srand48(seed); }
  inline void seed( long int seed ) {  srand48(seed); }
  inline double operator()() { return drand48() ; };

};
*/


//====================================
/** This class is an adaptor that turns a model of GSTL Random Number
Generator
 * into a model of STL Random Number Generator.
 * Objects of this class are conveniently created using helper function
 * STL_generator_adaptor.
 */

template <class GsTLGenerator>
class STL_generator_adaptor_t{
 public:
  typedef long int return_type;
  typedef long int argument_type;

  STL_generator_adaptor_t(GsTLGenerator& gen)
```

```
    : gen_(gen) {}

  inline return_type operator()(argument_type N) {
    return static_cast<return_type>(gen_()*N);
  }


 private:
  GsTLGenerator gen_;
};

// helper function
template <class GsTLGenerator>
inline STL_generator_adaptor_t<GsTLGenerator>
STL_generator_adaptor(GsTLGenerator& gen) {

  return STL_generator_adaptor_t<GsTLGenerator>(gen);
}

#endif
```

B.2.3 Quadrant Frequency Calculation Functions

```
//"quad_analysis.h" file
//This header file is used in quadrant frequency calculations.

#ifndef quad_analysis
#define quad_analysis

#include <iostream>
#include <sstream>
#include <string>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <iomanip>
#include <cmath>
#include "functions.h"
#include "read_histograms.h"
#include "quadrant.h"

using namespace std;

class MyOrig_Quad {
public:
      vector <int> QuadNum;
      vector<int> QuadFreq;
};

vector     <int>     QAnalysis(vector     <double>     Xs,vector<double>
Ys,vector<double>  Zs,  int  Qdx,int  Qdy,int  Qdz,  MyOrig_Quad  Orig_Quad,
vector <int> SNode, int FreqLim) {
      vector <int> New_Cand (0);
```

```
      vector <int> Ret_Cand;
      My_Quad MySim;
      MySim=Quadrant(Xs,Ys,Zs,Qdx,Qdy,Qdz);

      vector<int> QFreqDat;
      vector<int> QValDat;
      int fre;

      for (int ct=0; ct<MySim.QuadNum.size(); ct++) {
            if (findE(MySim.QuadNum.at(ct),Orig_Quad.QuadNum)==1){
                  fre=findI(MySim.QuadNum.at(ct),Orig_Quad.QuadNum);
                  if
(MySim.QNodes.at(ct).size()<Orig_Quad.QuadFreq.at(fre)) {

      New_Cand.insert(New_Cand.end(),MySim.QNodes.at(ct).begin(),MySim.
QNodes.at(ct).end());
                  }
            }
            else  if (MySim.QNodes.at(ct).size()<=FreqLim){

      New_Cand.insert(New_Cand.end(),MySim.QNodes.at(ct).begin(),MySim.
QNodes.at(ct).end());
            }
      }

      for (int ct=0; ct<New_Cand.size(); ct++) {
            if (findE(New_Cand.at(ct),SNode)==0) {
                  Ret_Cand.push_back(New_Cand.at(ct));
            }
      }
return Ret_Cand;
}
//============================================================//
#endif // quad_analysis

//"quadrant.h" file
//This header file is used by "quadrant_analysis.h" file

#ifndef quadrant
#define quadrant

#include <iostream>
#include <sstream>
#include <string>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <iomanip>
#include <cmath>
#include "functions.h"

using namespace std;

class My_Quad {
```

```
public:
      vector <int> QuadNum;
      vector <vector<int>> QNodes;
};

My_Quad Quadrant(vector<double> Xs,vector<double> Ys,vector<double> Zs,
int Qx, int Qy, int Qz) {
      int maxindX=findmaxD2(Xs);
      int maxindY=findmaxD2(Ys);
      int maxindZ=findmaxD2(Zs);

      My_Quad QuadVal;

      vector <int> Qs;
      vector <vector <int>> NNum;

      int minindX=findminD(Xs);
      int minindY=findminD(Ys);
      int minindZ=findminD(Zs);

      double XRange=(Xs.at(maxindX)-Xs.at(minindX))/Qx;
      double YRange=(Ys.at(maxindY)-Ys.at(minindY))/Qy;
      double ZRange=(Zs.at(maxindZ)-Zs.at(minindZ))/Qz;

      int nxx=1;
      int nyy=1;
      int nzz=1;
      int QV;

      for (int kl=0;kl<Xs.size();kl++) {
            if (XRange!=0) {
                  nxx=ceil((Xs.at(kl)-Xs.at(minindX))/XRange);
                  if (kl==maxindX) {
                  nxx=fabs(floor((Xs.at(kl)-Xs.at(minindX))/XRange));}

                  if (Xs.at(kl)==Xs.at(minindX)) {
                        nxx=1;}
            }

            if (YRange!=0) {
                  nyy=fabs(ceil((Ys.at(kl)-Ys.at(minindY))/YRange));
                  if (kl==maxindY) {
                  nyy=fabs(floor((Ys.at(kl)-Ys.at(minindY))/YRange));}
                  if (Ys.at(kl)==Ys.at(minindY)) {
                        nyy=1;}
            }

            if (ZRange!=0) {
                  nzz=fabs(ceil((Zs.at(kl)-Zs.at(minindZ))/ZRange));
                  if (kl==maxindZ) {
                  nzz=fabs(floor((Zs.at(kl)-Zs.at(minindZ))/ZRange));}
                  if (Zs.at(kl)==Zs.at(minindZ)) {
                        nzz=1;}
            }
```

```
            QV=nxx+(nyy-1)*Qx+(nzz-1)*Qx*Qy;

                  if (findE(QV,Qs)==0) {
                        Qs.push_back(QV);
                        NNum.push_back(vector <int> ());
                        NNum.back().push_back(kl);}
                  else {
                        int Qind=findI(QV,Qs);
                        NNum.at(Qind).push_back(kl);}
      }

      QuadVal.QuadNum.assign(Qs.begin(),Qs.end());
      QuadVal.QNodes.assign(NNum.begin(),NNum.end());

      return QuadVal;
}//End of Quadrant
//============================================================//
#endif // quadrant
```

B.2.4 Header and Function Files for Cave Thickness Simulation

```
//"thickness_sim.h" file used in Main Simulation cpp
#ifndef __thickness_sim_H__
#define __thickness_sim_H__

#include "thickness_sim.cpp"
#include "SimThickness.cpp"
#include "thickness_inter.cpp"

#endif
//End of thickness_sim.h file

//"thickness_sim_class.h" file
//This file is used in "thickness_sim.cpp"
//Variogram  model  used  in  thickness  modeling  is  defined  under
"covariance" class.
#ifndef __thickness_sim_class_h__
#define __thickness_sim_class_h__

class Point{
public:
  typedef double coordinate_type;
  Point(){X=0;Y=0;Z=0;};
  Point(double x, double y, double z): X(x), Y(y),Z(z) {};
  double& operator[](int i){
    if(i==1) return X;
      else if (i==2)  return Y;
      else if (i==3)  return Z;}

      const double& operator[](int i) const {
    if(i==1) return X;
    else if (i==2) return Y;
```

```
      else if (i==3)  return Z;}
  double X;
  double Y;
  double Z;
};

//-------------------------
class Node{
public:
  typedef double property_type;
  typedef Point location_type;

  Node():pval_(0) {loc_.X=0; loc_.Y=0;loc_.Z=0;};
  Node(double   x,   double   y,double   z,   double   pval){loc_.X=x;
loc_.Y=y;loc_.Z=z;pval_=pval;};
  inline Point location() const { return loc_;};
  inline double& property_value(){return pval_;}
  inline const double& property_value() const {return pval_;}
private:
  Point loc_;
  double pval_;};
//---------------------------
inline Node NodeC (double x,double y, double z, double pval) {
      Node MyNode(x,y,z,pval);
      return MyNode;}
//---------------------------
inline Point PointC (double x,double y, double z) {
      Point MyPoint(x,y,z);
      return MyPoint;}
//---------------------------

class covariance{
public:

  inline double operator()(const Point& P1,const Point& P2) const {
    double                   h=sqrt(double(pow((P1.X-P2.X),2)+pow((P1.Y-
P2.Y),2)+pow((P1.Z-P2.Z),2)));
      double hx=fabs(P1.X-P2.X);
      double hy=fabs(P1.Y-P2.Y);
      double hz=fabs(P1.Z-P2.Z);
      double pi_a= 3.14159;

      //Define properties for anisotropic variogram
      //double VarAzm, VarDip; //Variogram properties
      //VarAzm=(45*pi_a)/180; //Azimuth Angle
      //VarDip=(5*pi_a)/180; //Dip Angle
      //double hxx=cos(VarDip)*sin(VarAzm)*hx
                  +cos(VarDip)*cos(VarAzm)*hy -sin(VarAzm)*hz;
      //double hyy=-cos(VarAzm)*hx+sin(VarAzm)*hy;
      //double hzz=sin(VarDip)*sin(VarAzm)*hx
                  +sin(VarDip)*cos(VarAzm)*hy+cos(VarDip)*hz;
      //double h=sqrt(pow((hxx/900),2)+pow((hyy/900),2)
                 +pow((hzz/525),2)); //First Structure
```

```
      double  h=sqrt(pow((hx,2)+pow((hy,2)+pow((hz,2));  //For  isotropic
variogram
      double a=2408;
      double c=1;//sill contribution

    return c*(1-1.5*h/a+0.5*pow(h,3)/pow(a,3)); //Covariance
calculation method for the corresponding variogram model is defined at
this line.
  }
};

#endif


//"thickness_sim.cpp" file
//This file is used for thickness simulation. CalThick function returns
kriging estimate and estimation variance for cave thickness.

//____Kriging system solver, Gauss_Elim function from________
//  Programmed by Jun Ni, Ph.D.M.E., at jni@hpcsoft.com
//  Last modified July 17, 2003
//   Reference: Numerical Methods for Engineers by Griffiths and Smith,
1991, CRC Press
//______Matrix definitions from________
// GsTL: the Geostatistics Template Library
// Author: Nicolas Remy
// Copyright (c) 2000 The Board of Trustees of the Leland Stanford
Junior University
// All rights reserved.

#include <iostream>
#include <math.h>
#include <vector>
#include <algorithm>

using namespace std;

#include "thickness_sim_class.h"
//_____________________________________________________________________
template <class Ctr,class ConD,class NodeD>
vector <double> CalThick(Ctr center,ConD CondData,NodeD Nodes) {

std::vector<double> estimate (2);
covariance covar;
 std::vector<double> RHS; //(CondData.size()+1);

double c_h0;
int ct,ctt;

Symmetric_matrix <double> LHS (CondData.size()+1);

for (ct=0; ct<CondData.size(); ct++) {
  LHS(ct+1,ct+1)=covar(CondData.at(ct),CondData.at(ct));
  RHS.push_back(covar(CondData.at(ct),center));
```

```
  for (ctt=0; ctt<CondData.size(); ctt++) {
        if (ct!=ctt) {
            LHS(ct+1,ctt+1)=covar(CondData.at(ct),CondData.at(ctt));
            LHS(ctt+1,ct+1)=LHS(ct+1,ctt+1);}
  }
}

for (ct=0; ct<CondData.size(); ct++) {
  LHS(CondData.size()+1,ct+1)=1;
  LHS(ct+1,CondData.size()+1)=1;}

LHS(CondData.size()+1,CondData.size()+1)=0;
RHS.push_back(1);

long double* Weig=Gauss_Elim(LHS,RHS); //Solve Kriging System
double est=0,wesum=0,varterm=0;
c_h0=1;

for (ct=0; ct<CondData.size(); ct++) {
      est=est+Weig[ct]*Nodes.at(ct).property_value();
      wesum=wesum+Weig[ct];
      varterm=varterm+Weig[ct]*RHS.at(ct+1);}
wesum=wesum+Weig[ct];

estimate.at(0)=est;
estimate.at(1)=double(c_h0-varterm-Weig[ct]);

if (estimate.at(1)>1) { estimate.at(1)=1; }

cout<<"Estimate="<<estimate[0]<<endl;
cout<<"Estimation Variance="<<estimate[1]<<endl;

return estimate; //Returns kriging estimate and estimation variance
} //End of CalThick


//"SimThickness.cpp" file
//This  file  is  used  in  cave  thickness  modeling.  It  returns  the
simulated thickness value.

#include <stdio.h>
#include <stdlib.h>
#include <vector>

#include "Gaussian_Elim.h"
#include "CDF_Draw.h"
#include "utils.h"
//____Kriging system solver, Gauss_Elim function from________
//  Programmed by Jun Ni, Ph.D.M.E., at jni@hpcsoft.com
//  Last modified July 17, 2003
//  Reference: Numerical Methods for Engineers by Griffiths and Smith,
//1991, CRC Press
```

```
//______CDF_Draw, utils and random number generator from________
// GsTL: the Geostatistics Template Library
// Author: Nicolas Remy
// Copyright (c) 2000 The Board of Trustees of the Leland Stanford
//Junior University
// All rights reserved.

using namespace std;
template <class Xs,class Ys, class Zs, class NConnec,class id,class
PVals, class VR>
vector <double> SimThick(int Type,Xs XVals, Ys YVals, Zs ZVals,NConnec
NCons,id aidx,PVals ThickV, const VR VarRange, int MaxN,ofstream&
MyfileN) {
      std:vector <Point> CondData;
      std::vector<Node> Nodes;
      std::vector<double> LagVals (1);
      Point Center(XVals.back(), YVals.back(),ZVals.back());

      if (Type==1) { //Calculate Thickness for Initialization
            for (int yy=0; yy<XVals.size()-1; yy++) {
                  double
Lag=Cal_h(Center.X,Center.Y,Center.Z,XVals.at(yy),YVals.at(yy),ZVals.at
(yy));
                  if (Lag<=VarRange) {
                        if (findED(Lag,LagVals)==0) {
                        LagVals.push_back(Lag);
      CondData.push_back(PointC(XVals.at(yy),YVals.at(yy),ZVals.at(yy))
);
      Nodes.push_back(NodeC(XVals.at(yy),YVals.at(yy),ZVals.at(yy),Thic
kV.at(yy)));            }
                  }
                  if (Nodes.size()>=MaxN) {
                        break;}
            }
      } //End of Calculate Thickness for Initialization

      if (Type==2) { //Calculate Thickness at Simulation Nodes
            for (int yy=0; yy<NCons.at(aidx).size()-1; yy++) {
                  int nid=NCons.at(aidx).at(yy);
                  double checkT=ThickV.at(nid);

                  if (checkT!=0) {
                        double
Lag=Cal_h(Center.X,Center.Y,Center.Z,XVals.at(nid),YVals.at(nid),ZVals.
at(nid));
                        if (Lag<=VarRange) {
                              if (findED(Lag,LagVals)==0) {
                              LagVals.push_back(Lag);


      CondData.push_back(PointC(XVals.at(nid),YVals.at(nid),ZVals.at(ni
d)));
```

```
      Nodes.push_back(NodeC(XVals.at(nid),YVals.at(nid),ZVals.at(nid),T
hickV.at(nid)));              }
                        }
                  }

                  if (Nodes.size()>=MaxN) {
                        break;}
            }
      }//End of Calculate Thickness at Simulation Nodes
      vector <double> Th;

      if (CondData.size()!=0) { //There are cond.data points within the
range
            Th=CalThick(Center,CondData,Nodes);} //Performing Ordinary
Kriging
      else {
            double ThSum=sumall(ThickV);
            Th.push_back(ThSum/ThickV.size());
            double *M=&Th.at(0);
            Th.push_back(variance(ThickV.begin(),ThickV.end(),M));}

      double pV;
      pV=rand_generator_();
      while (pV>=0.9){
            pV=rand_generator_(); }

      double TData;
      TData=DrawVal(Th.at(0),Th.at(1),pV); //Monte Carlo Sample from a
Gaussian Distribution with mean=Ok_estimate and variance=Ok_variance

      MyfileN<<Center.X<<" "<<Center.Y<<" "<<Center.Z<<" "<<TData<<
      " "<<Th.at(1)<<endl;
      std::cout<<"Thick="<<TData<<std::endl;
      Th.at(0)=TData;
      return Th;
}//End of SimThick

//End of SimThickness.cpp file

//"thickness_inter.h" header file
//Cave Thickness Interpolation header file
#ifndef __thickness_inter_h__
#define __thickness_inter_h__

#include "thickness_inter.cpp"
#endif


//"thickness_inter.cpp" file
//This function performs linear interpolation of cave thickness between
two given nodes.
#include <iostream>
#include <math.h>
```

```
#include <vector>
#include <algorithm>

using namespace std;

template<class DCoor, class TVal>
vector <double> ThickInt(int DiscStep,int LineN,DCoor X1,DCoor Y1,DCoor
Z1,TVal  Th1,TVal  Var1,DCoor  X2,DCoor  Y2,DCoor  Z2,TVal  Th2,TVal  Var2,
ofstream& MyfileN, ofstream& MyfileN2) {
      vector <double> RetVals (4);
      vector <double> RVs (DiscStep);
      vector <double> ThV (DiscStep);
      double MyD=Cal_h(X1,Y1,Z1,X2,Y2,Z2);

      RVs.at(0)=rand_generator_();//(rand()/double(RAND_MAX+1))*(0.5);
      for (int ii=1; ii<DiscStep; ii++) {

      RVs.at(ii)=rand_generator_();//(rand()/double(RAND_MAX+1))*(0.5);
            while (abs(RVs.at(ii)-RVs.at(0))<(1/DiscStep)-0.1){

      RVs.at(ii)=rand_generator_();//(rand()/double(RAND_MAX+1))*(0.5);

            }

      }

      for (int ii=0; ii<DiscStep; ii++) {
            double rv=RVs.at(ii);//(rand()/double(RAND_MAX+1))*(0.5+1);
            double sv=1-rv;

            rv=rv*MyD;
            sv=sv*MyD;

            RetVals.at(0)=(rv*X2+sv*X1)/(rv+sv);
            RetVals.at(1)=(rv*Y2+sv*Y1)/(rv+sv);
            RetVals.at(2)=(rv*Z2+sv*Z1)/(rv+sv);

            double ThDif=Th1-Th2;
            double VarDif=abs(Var1-Var2);

            RetVals.at(3)=Th1-(rv*ThDif)/MyD;
            double Var_Intr=min(Var1,Var2)+(rv*VarDif)/MyD;

            ThV.at(ii)=RetVals.at(3);
            if (Th1!=0 || Th2!=0) {

                  MyfileN<<LineN<<" "<<RetVals.at(0)<<" "
      <<RetVals.at(1)<<" "<<RetVals.at(2)<<" "<<RetVals.at(3)<<endl;
                  MyfileN2<<RetVals.at(0)<<" "<<RetVals.at(1)<<" "
      <<RetVals.at(2)<<" "<<RetVals.at(3)<<" "<<Var_Intr<<endl;
            }
      }
return ThV;
} //End of ThickInt
```

B.2.5. Pattern Simulation Main Program

```
//========Pattern Simulation Main Program==============//
//This program is developed by Selin Erzeybek Balan, UT-Austin.
//Please include this information and acknowledge the author by proper
citations.

//Please check the input file formats.
//Enter the input file names, input parameters and make necessary
adjustments before running.
//For random number generator additional header files, please refer to
GsTL:Geostatistical Template Library of SCRF, Stanford University.

#include <iostream>
#include <fstream>
#include <iomanip>
#include <sstream>
#include <cmath>
#include <vector>
#include <bitset>
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include <algorithm>
#include <time.h>
#include <valarray>

#include "functions.h"
#include "read_templates.h"
#include "read_histograms.h"
#include "read_patterns.h"
#include "histogram_dist.h"
#include "quad_analysis.h"
#include "thickness_sim.h"
#include "globals.h"

using namespace std;

//==========Parameters and Constants============//
//These values can also be read from an input file or command window if
implemented.
int nrow=145;//Size of input file (number of rows)
int n=145, nline=1; //Number of conditioning data; nline:connection
number
int inc_max=500; //Maximum simmulation iteration step
double DCond=100;//Minimum separation between the conditioning data
int ScanStep=75;//Histogram deviation control step
int checkstep=50;//Cluster node frequency control step
int QuadLim=10; //Max. node frequency for clusters with zero-frequency
originally
int DataRead=0; //Data Reading Criteria: 0=Sequential Reading; 1=Random
Selection;
```

```
//These parameters are for cave opening simulation. If cave opening
simulation is not performed, comment these parameters
int NSeg=3; //Number of discretezation segments in thickness
interpolation along a line
double VRange=2408; //Variogram Range for cave opening simulation
int MaxNumD=100; //Maximum number of data to be used in kriging
int DataType=2; //Thickness distribution type: DataType=1; Normal
Distribution;  DataType=2; Log-Normal Distribution
//==============================

//Tolerance Window Definition===============
//These values can also be read from an input file or command window if
implemented.
double Azm_Tol=20; //Azimuth tolerance
double Dip_Tol=6; //Dip tolerance
double Dist_Tol=0.30; //Lag tolerance factor: Lag tolerance= Lag *
Dist_Tol
double pi = 3.14159;
int BinTDec;
//=============================================//

//==================Input Files==================//
ifstream myfile1("input_files.dat");//Input file for
Templates,Histograms and Pattern Configurations
ifstream infile("sample_data.dat");//Conditioning data file
ifstream infile3("highest_freq_pattern.dat"); //Result of MPS
algorithm: Pattern with the Max. Connections & Highest Frequency

//==Optional Files===//

//If quadrant information is available
ifstream infile4("quadrant_updated.dat"); //Quadrant Number and
Frequency
ifstream infile5("quadrant_prop.dat"); //Quadrant properties

//If cave opening simulation is performed
ifstream infile7("thickness.dat");//Thickness values at the
conditioning data

//If surface dip map is available
ifstream infile8("gocad_dip_map.dat");//Dip map by gocad

//==============Output Files====================//

ofstream outfile("connections_sample_data.dat"); //Stores simulated
connections in a line stick format
ofstream outfile2("nodes_connec_sample_data.dat"); //Indices of the
nodes connected
ofstream outfile3("cond_data_sample_data.dat"); //Conditioning data
ofstream outfile4("locations_sample_data.dat"); //Simulated node
coordinates including cave and non-cave locations
ofstream outfile5("connected_sample_data.dat"); //Simulated cave node
locations
```

```
ofstream outfile6("selectedpatterns_sample_data_deneme.dat");
//Patterns retreived from the pattern histogram
ofstream outfile7("petrel_connec_sample_data_deneme.txt"); //Stores
simulated connections in Petrel general line format
ofstream outfile8("mothernode_sample_data_deneme.txt"); //Stores
"mother nodes" of the simulated caves and flag
ofstream outfile9("azmAngles_sample_data_deneme.dat"); //Stores azimuth
angles of the simulated cave passage
ofstream outfile10("thickness_sample_data_deneme.dat"); //Thickness
values at the simulated nodes
ofstream outfile11("thickness_map_sample_data_deneme.dat"); //Thickness
values at the simulated nodes and the discretized nodes of the passages
ofstream outfile12("transform_sample_data_deneme.dat"); //Simulated
thickness values and the estimation variances for data transformation
if needed.

vector <int> TInd;

int* Index= new int [];
int i;
double td;

//**************========MAIN PROGRAM==========***************//
int main()
{
//Seed number for random number generator
rand_generator_.seed(1000);

vector<double>XLoc (n);
vector<double>YLoc (n);
vector<double>ZLoc (n);
vector<double>XNLoc;
vector<double>YNLoc;
vector<double>ZNLoc;
double **Data;
Data=new double*[nrow];
double **ConDat;
ConDat=new double*[n];

vector<vector <double> > DelX(n, vector<double> (n ));
vector<vector <double> > DelY(n ,vector<double> (n ));
vector<vector <double> > DelZ(n, vector<double> (n ));
vector<vector <double> > D1(n ,vector<double> (n ));
vector<vector <double> > Phi(n, vector<double> (n ));
vector<vector <double> > h(n, vector<double> (n ));
vector<vector <double> > Theta(n, vector<double> (n ));

vector <int> FlagL; //Store node flag: 1=cave; 0=non-cave
vector<int>SimNode; //Store "simulation node" indices
vector<int>CandNode; //Store "simulatable node" indices
vector<vector<int>> NodeCon (n, vector<int> ()); //store mother-
daughter node indices
vector<vector <double> > NodeConLoc; //store coordinates of connected
nodes
```

```
vector<int> MotherN ;

//====Use with Cave Opening Modeling===
vector<double> EstVar; //Estimation Variance
vector <double> ThRes (2); //Thickness and EstVr values

static const unsigned BIT_LENGTH = 27;

//=====Loading the Input Data======//
for (i=0; i<nrow; i++) {
      Data[i]=new double[3];
      for (int j=0; j<3; j++) {
      infile>>Data[i][j];
      }
}

//====Loading the Templates, Histograms and Patterns======//

//Load Templates
vector <My_Temp> ReadTemp;
ReadTemp=Read_Temp(myfile1);

myfile1.clear();
myfile1.seekg(0, ios::beg);

//Load Histograms
vector <My_Hist> ReadHist;
ReadHist=Read_Hist(myfile1);

vector <vector <double>> MyDist(ReadHist.size(), vector <double> ());

myfile1.clear();
myfile1.seekg(0, ios::beg);

//Load Patterns (temp_node_files)
vector <My_TNode> ReadPat;
ReadPat=Read_Pat(myfile1);

myfile1.clear();
myfile1.seekg(0, ios::beg);

int NTemp; //Number of Templates to be used
myfile1>>NTemp;


//=====Determine Data Boundary (Minimum and Maximum X,Y,Z Coordinates
of Data)=====//
vector <double> MinBndCoor (3); //[X,Y,Z]; Minimum Boundary Coordinates
vector <double> MaxBndCoor (3); //[X,Y,Z]; Maximum Boundary Coordinates

//Use "bounds" to get the boundaries if data limits will be used.
Otherwise, enter the boundary limits manually.
MinBndCoor=bounds(Data,1,nrow); //(Array,Select) Select=1 for min,
Select=2 for max.
```

```
MaxBndCoor=bounds(Data,2,nrow);

//===If input data limits are different than the boundary, enter
thelimits manually
//MinBndCoor.at(0)=1494046.14; //1494460;
//MinBndCoor.at(1)=435040.37;//440520;
//MinBndCoor.at(2)=825; //966.59;//1155;

//MaxBndCoor.at(0)=1511575.05;//1504280;
//MaxBndCoor.at(1)=458755.95;//1452877;
//MaxBndCoor.at(2)=1479.19;//1476.41;//1483;

//============Loading the Highest Frequency Pattern============//
double a;
int num;
cout<<"Enter the total number of nodes in the highest frequency the
pattern (# of rows in highfreq_xx.dat):"<<" ";
cin>>num;
vector <vector<double>> Pattern;
i=0;
int j=0;
for(j=0;j<num; j++ ){
      Pattern.push_back(vector <double> ());
      infile3>>a;
      Pattern[j].push_back(a);//Azimuth Angle
      infile3>>a;
      Pattern[j].push_back(a);//Dip Angle
      infile3>>a;
      Pattern[j].push_back(a);//Distance
}

//============Load Quadrant Frequency and Properties================//
MyOrig_Quad Orig_Quadrant;
string my_qline;
getline(infile4,my_qline);    //Read Header line
while (!infile4.eof( )){
      infile4>>a;
      Orig_Quadrant.QuadNum.push_back(a);
      infile4>>a;
      Orig_Quadrant.QuadFreq.push_back(a);
}

Orig_Quadrant.QuadNum.pop_back();
Orig_Quadrant.QuadFreq.pop_back();

vector <double> Orig_Range(3);
vector <int> Quad_Num (3);

for(j=0;j<5; j++ ){
getline(infile5,my_qline); } //Read header and other lines

for(j=0;j<3; j++ ){
      infile5>>Orig_Range.at(j); }
```

```
getline(infile5,my_qline); //Read header and other lines
getline(infile5,my_qline); //Read header and other lines

for(j=0;j<3; j++ ){
      infile5>>Quad_Num.at(j); }

//=================Load Dip Map Generated by GOCAD=======
//If dip map is used
vector<vector <double> > DipMap=read_dip_map(infile8);

//==========Randomly Gather Conditioning Locations================//
int low=0, high=nrow-1;
int fr;
vector <int> rand_loc (n);
vector <double> thickness (n);

//Conditioning Data is imported sequentially
if (DataRead==0) {
            for (i=0; i<n; i++) {
                  XLoc[i]=Data[i][0];
                  YLoc[i]=Data[i][1];
                  ZLoc[i]=Data[i][2];

                  infile7>>thickness.at(i);

                  SimNode.push_back(i);
                  FlagL.push_back(1);

                  NodeConLoc.push_back(vector <double> (3));
                  NodeConLoc.back().at(0)=XLoc[i];
                  NodeConLoc.back().at(1)=YLoc[i];
                  NodeConLoc.back().at(2)=ZLoc[i];

                  outfile3<<XLoc.at(i)<<" "
                        <<YLoc.at(i)<<" "
                        <<ZLoc.at(i)<<endl;

                  outfile4<<XLoc.at(i)<<" "
                        <<YLoc.at(i)<<" "
                        <<ZLoc.at(i)<<endl;

                  outfile5<<XLoc.at(i)<<" "
                        <<YLoc.at(i)<<" "
                        <<ZLoc.at(i)<<endl;

             outfile10<<XLoc.at(i)<<" "<<YLoc.at(i)<<" "<<ZLoc.at(i)
                        <<" "<<thickness.at(i)<<endl;
            }
}

//=============================================//
//Conditioning Data is imported randomly
if (DataRead==1) {
            for (i=0; i<n; i++) {
```

```
        if (i==0) {
              infile7>>thickness.at(i);

              rand_loc.at(i)=low+
                          (high-low+1)*rand_generator_();
                          //rand()/(RAND_MAX + 1.0));

              int ranv=low+(high-low+1)*rand_generator_();
              // rand()/( (RAND_MAX + 1.0));

              fr=findE(ranv,rand_loc);

              if (i!=0 && fr==1) {
                    while (findE(ranv,rand_loc)==1) {

                    ranv=low+(high-low+1)*rand_generator_();
                    }
              }

              rand_loc.at(i)=ranv;
              XLoc.at(i)=Data[rand_loc.at(i)][0];
              YLoc.at(i)=Data[rand_loc.at(i)][1];
              ZLoc.at(i)=Data[rand_loc.at(i)][2];

              infile7>>thickness.at(i);

              outfile3<<XLoc.at(i)<<" "
                    <<YLoc.at(i)<<" "
                    <<ZLoc.at(i)<<endl;

              outfile4<<XLoc.at(i)<<" "
                    <<YLoc.at(i)<<" "
                    <<ZLoc.at(i)<<endl;

              outfile5<<XLoc.at(i)<<" "
                    <<YLoc.at(i)<<" "
                    <<ZLoc.at(i)<<endl;

              outfile10<<XLoc.at(i)<<" "<<YLoc.at(i)<<" "
                    <<ZLoc.at(i)<<" "<<thickness.at(i)<<endl;

              NodeConLoc.push_back(vector <double> (3));
              NodeConLoc.back().at(0)=XLoc.at(i);
              NodeConLoc.back().at(1)=YLoc.at(i);
              NodeConLoc.back().at(2)=ZLoc.at(i);

              SimNode.push_back(i);
              FlagL.push_back(1);
        }
        else {
              rand_loc.at(i)=low+(high-
                                low+1)*rand_generator_();
//low+((high-low+1)*rand()/(RAND_MAX + 1.0)); // if srand()
is used;
```

```
                        int ranv=low+(high-low+1)*rand_generator_();
            //low+((high-low+1)*rand()/(RAND_MAX + 1.0)); // if srand()
      is used;
                        fr=findE(ranv,rand_loc);

                        if (i!=0 && fr==1) {
                              while (findE(ranv,rand_loc)==1) {

                                    ranv=low+(high-
                                          low+1)*rand_generator_();
            //low+((high-low+1)*rand()/(RAND_MAX + 1.0));// if srand()
            is used;
                              }
                        }
                        rand_loc.at(i)=ranv;
                        vector<double> DLoc;
                        for (int km=0; km<XLoc.size(); km++) {
      DLoc.push_back(Cal_h(XLoc.at(km),YLoc.at(km),ZLoc.at(km),Data[ran
d_loc.at(i)][0],Data[rand_loc.at(i)][1],Data[rand_loc.at(i)][2]));
                        }

                        vector <int> Mykm;
                        Mykm=findany(DLoc,DCond,1);

                        while (Mykm.empty()==0) {
                        rand_loc.at(i)=low+(high-
                                          low+1)*rand_generator_();
      //low+((high-low+1)*rand()/(RAND_MAX + 1.0)); // if srand() is
used;
                        int ranv=low+(high-low+1)*rand_generator_();
      //low+((high-low+1)*rand()/(RAND_MAX + 1.0)); // if srand() is
used;
                              fr=findE(ranv,rand_loc);

                              if (i!=0 && fr==1) {
                                    while (findE(ranv,rand_loc)==1) {

                                    ranv=low+(high-
                                    low+1)*rand_generator_();
                              //low+((high-low+1)*rand()/(RAND_MAX +
                              1.0)); // if srand() is used;
                                    }
                              }

                              rand_loc.at(i)=ranv;
                              vector<double> DLoc;

                              for (int km=0; km<XLoc.size(); km++) {

      DLoc.push_back(Cal_h(XLoc.at(km),YLoc.at(km),ZLoc.at(km),Data[ran
d_loc.at(i)][0],Data[rand_loc.at(i)][1],Data[rand_loc.at(i)][2]));
                              }
                              Mykm=findany(DLoc,DCond,1);
```

```
                        } //End of while (Mykm.empty()==1)

                        XLoc.at(i)=Data[rand_loc.at(i)][0];
                        YLoc.at(i)=Data[rand_loc.at(i)][1];
                        ZLoc.at(i)=Data[rand_loc.at(i)][2];

                        infile7>>thickness.at(i);

                        NodeConLoc.push_back(vector <double> (3));
                        NodeConLoc.back().at(0)=XLoc.at(i);
                        NodeConLoc.back().at(1)=YLoc.at(i);
                        NodeConLoc.back().at(2)=ZLoc.at(i);

                        SimNode.push_back(i);
                        FlagL.push_back(1);

                        outfile3<<XLoc.at(i)<<" "
                                    <<YLoc.at(i)<<" "
                                    <<ZLoc.at(i)<<endl;

                        outfile4<<XLoc.at(i)<<" "
                                    <<YLoc.at(i)<<" "
                                    <<ZLoc.at(i)<<endl;

                        outfile5<<XLoc.at(i)<<" "
                                    <<YLoc.at(i)<<" "
                                    <<ZLoc.at(i)<<endl;

                        outfile10<<XLoc.at(i)<<" "<<YLoc.at(i)<<" "
                                    <<ZLoc.at(i)<<"
                                "<<thickness.at(i)<<endl;
                    }// End of else if (i==0)
                }
        }//End of switch

        //Data Transformation (for Log-Normal Data Sets)
        if (DataType==2) {//Thickness data is Log-Normal.Convert to
Normal Data.
                thickness=CalLog(thickness);
        }

        for (i=0; i<thickness.size(); i++) {
                    EstVar.push_back(0);
                    outfile12<<XLoc.at(i)<<" "<<YLoc.at(i)<<" "
                        <<ZLoc.at(i)<<" "
                        <<thickness.at(i)<<" "<<0<<endl;
    }

//======Apply the Highest Frequency Pattern at the Conditioning Data
Locations====//
int conc=1;
int rnd;
double Dipp;
vector <double> TempCord2 (3);
```

```
vector <double> TempCord3 (3);
vector <double> AzmAngVals;
vector <double> TInter;
double* TempCord;
for (i=0; i<n; i++) { //Loop over Conditioning Data Locs

      rnd=(num+1)*rand_generator_();//rand() % (num+1); //
num=Pattern.size();
      //cout<<i<<" "<<rnd<<endl;

      for (int jj=0; jj<Pattern.size(); jj++) { //Loop over Pattern

            //If Dip map is used

      Dipp=find_temp_dip(XLoc.at(i),YLoc.at(i),ZLoc.at(i),DipMap);

            if (rnd!=jj) { //Pattern.size()!=1
                  if (rnd!=num) { //Pattern.size()
                        //If Dip map is used, Pattern[jj][1] is
replaced by Dipp

      TempCord=PatBackCal2(rnd,jj,Pattern[jj][0],Dipp,Pattern[jj][2],XL
oc.at(i),YLoc.at(i),ZLoc.at(i),Pattern[rnd][0],Pattern[rnd][1],Pattern[
rnd][2],Dist_Tol,Azm_Tol,Dip_Tol);
                        TempCord2.at(0)=TempCord[0];
                        TempCord2.at(1)=TempCord[1];
                        TempCord2.at(2)=TempCord[2];
                        }
                  else {
                        //Cond. Data is at the center
                        //If Dip map is used, Pattern[jj][1] is
replaced by Dipp

      TempCord=BackCal2(Pattern[jj][0],Dipp,Pattern[jj][2],XLoc.at(i),Y
Loc.at(i),ZLoc.at(i),Dist_Tol,Azm_Tol,Dip_Tol);
                        TempCord2.at(0)=TempCord[0];
                        TempCord2.at(1)=TempCord[1];
                        TempCord2.at(2)=TempCord[2];
                        } //if (rnd<Pattern.size()) {

                  if (BndContl(TempCord2,MinBndCoor, MaxBndCoor)==1) {
                  //New node is within the boundaries
                        XLoc.push_back(TempCord[0]);
                        YLoc.push_back(TempCord[1]);
                        ZLoc.push_back(TempCord[2]);

                        outfile4<<TempCord[0]<<" "
                              <<TempCord[1]<<" "
                              <<TempCord[2]<<endl;

                        outfile5<<TempCord[0]<<" "
                              <<TempCord[1]<<" "
                              <<TempCord[2]<<endl;
```

```
                        CandNode.push_back(XLoc.size()-1);
                        NodeCon.at(i).push_back(XLoc.size()-1);

                        MotherN.push_back(i);
                        FlagL.push_back(1);

                        NodeConLoc.push_back(vector <double> (3));
                        NodeConLoc.back().at(0)=XLoc.back();
                        NodeConLoc.back().at(1)=YLoc.back();
                        NodeConLoc.back().at(2)=ZLoc.back();

                        outfile8<<MotherN.back()<<" "
                              <<FlagL.back()<<endl;

                        //Calculate Thickness for the new node
                        ThRes=SimThick(1,XLoc,YLoc,ZLoc, NodeCon,
                        MotherN.back(),thickness,VRange, MaxNumD,
                        outfile12);
                        thickness.push_back(ThRes.at(0));
                        EstVar.push_back(ThRes.at(1));

                        outfile10<<XLoc.back()<<" "<<YLoc.back()<<" "
                        <<ZLoc.back()<<" "<<thickness.back()<<endl;
                        outfile11<<nline<<" "<<XLoc.back()<<" "
                        <<YLoc.back()<<" "<<ZLoc.back()<<" "
                        <<thickness.back()<<endl;

TInter=ThickInt(NSeg,nline,XLoc.back(),YLoc.back(),ZLoc.back(),thicknes
s.back(),EstVar.back(),XLoc.at(i),YLoc.at(i),ZLoc.at(i),thickness.at(i)
,EstVar.at(i),outfile11,outfile12);
                        outfile11<<nline<<" "<<XLoc.at(i)<<" "
                              <<YLoc.at(i)<<" "
                              <<ZLoc.at(i)<<" "<<thickness.at(i)<<endl;

                        nline=nline+1;
                        if (rnd==num) { //Pattern.size()
                              AzmAngVals.push_back(TempCord[3]);

                              //Cond. Data is at the center
                              outfile<<conc<<" "<<XLoc.at(i)<<" "
                              <<YLoc.at(i)<<" "<<ZLoc.at(i)<<endl;
                              outfile<<conc<<" "<<TempCord[0]<<" "
                              <<TempCord[1]<<" "<<TempCord[2]<<endl;

                              outfile7<<XLoc.at(i)<<" "
                              <<YLoc.at(i)<<" "<<ZLoc.at(i)<<endl;
                              outfile7<<TempCord[0]<<" "
                              <<TempCord[1]<<" " <<TempCord[2]<<endl;
                              outfile7<<-999<<" "<<-999<<" "
                                    <<-999<<endl;

                              conc++;
                        }
                        else
```

```
                        {     TempCord3.at(0)=TempCord[3];
                              TempCord3.at(1)=TempCord[4];
                              TempCord3.at(2)=TempCord[5];

                              AzmAngVals.push_back(TempCord[6]);

                              if (BndContl(TempCord3,MinBndCoor,
MaxBndCoor)==1) {

                        //Store Center Node to New Node Connection
                              outfile<<conc<<" "<<TempCord[3]<<" "
                              <<TempCord[4]<<" "<<TempCord[5]<<endl;
                                    outfile<<conc<<" "
                                          <<TempCord[0]<<" "
                                          <<TempCord[1]<<" "
                                          <<TempCord[2]<<endl;
                                    conc++;

                              outfile7<<TempCord[3]<<" "
                              <<TempCord[4]<<" "<<TempCord[5]<<endl;
                              outfile7<<TempCord[0]<<" "
                              <<TempCord[1]<<" "<<TempCord[2]<<endl;
                              outfile7<<-999<<" "<<-999<<" "
                                    <<-999<<endl;
                              }

                        }
                  }
            }
            else {
                        if (rnd<num) { //Pattern.size()

      TempCord=PatBackCal(rnd,jj,Pattern[jj][0],Pattern[jj][1],Pattern[
jj][2],XLoc.at(i),YLoc.at(i),ZLoc.at(i),Pattern[rnd][0],Pattern[rnd][1]
,Pattern[rnd][2]);
                              TempCord2.at(0)=TempCord[3];
                              TempCord2.at(1)=TempCord[4];
                              TempCord2.at(2)=TempCord[5];
                        }
                        else {
                              //Cond. Data is at the center

      TempCord=BackCal(Pattern[jj][0],Pattern[jj][1],Pattern[jj][2],XLo
c.at(i),YLoc.at(i),ZLoc.at(i));
                              TempCord2.at(0)=TempCord[0];
                              TempCord2.at(1)=TempCord[1];
                              TempCord2.at(2)=TempCord[2];
                        }

                        if (BndContl(TempCord2,MinBndCoor,
MaxBndCoor)==1) { //New node is within the boundaries
                              XLoc.push_back(TempCord2.at(0));
                              YLoc.push_back(TempCord2.at(1));
```

```
                              ZLoc.push_back(TempCord2.at(2));

                              CandNode.push_back(XLoc.size()-1);
                              FlagL.push_back(1);

                              //Store the connections
                              outfile<<conc<<" "<<XLoc.at(i)<<" "
                              <<YLoc.at(i)<<" "<<ZLoc.at(i)<<endl;

                              outfile<<conc<<" "<<TempCord2.at(0)<<" "
                        <<TempCord2.at(1)<<" "<<TempCord2.at(2)<<endl;

                              outfile7<<XLoc.at(i)<<" "
                              <<YLoc.at(i)<<" "
                              <<ZLoc.at(i)<<endl;
                              outfile7<<TempCord2.at(0)<<" " <
                              <TempCord2.at(1)<<" "
                              <<TempCord2.at(2)<<endl;
                              outfile7<<-999<<" "<<-999<<" "
                                    <<-999<<endl;

                              NodeCon.at(i).push_back(XLoc.size()-1);

                              MotherN.push_back(i);

                              outfile4<<TempCord2.at(0)<<" "
                                    <<TempCord2.at(1)<<" "
                                    <<TempCord2.at(2)<<endl;

                              outfile5<<TempCord2.at(0)<<" "
                                    <<TempCord2.at(1)<<" "
                                    <<TempCord2.at(2)<<endl;

                              ThRes=SimThick(1,XLoc,YLoc,ZLoc,
          NodeCon,MotherN.back(),thickness,VRange, MaxNumD, outfile12);

                              thickness.push_back(ThRes.at(0));
                              EstVar.push_back(ThRes.at(1));

                              outfile10<<XLoc.back()<<" "
                                    <<YLoc.back()<<" "
                                    <<ZLoc.back()<<" "
                                    <<thickness.back()<<endl;
                              outfile11<<nline<<" "<<XLoc.back()<<" "
                                          <<YLoc.back()<<" "
                                          <<ZLoc.back()<<" "
                                          <<thickness.back()<<endl;

      TInter=ThickInt(NSeg,nline,XLoc.back(),YLoc.back(),ZLoc.back(),th
ickness.back(),EstVar.back(),XLoc.at(i),YLoc.at(i),ZLoc.at(i),thickness
.at(i),EstVar.at(i),outfile11,outfile12);
                              outfile11<<nline<<" "<<XLoc.at(i)<<" "
                                    <<YLoc.at(i)<<" "<<ZLoc.at(i)<<" "
                                    <<thickness.at(i)<<endl;
```

```
                              nline=nline+1;
                              conc++;
                         }
                    } //end of Pattern.size()==1
      } //End of for (j=0; j<Pattern.size(); j++)

      //Store Center Node to Cond. Data Connection if applicable
      if (rnd<num) { //Pattern.size()
            if (BndContl(TempCord3,MinBndCoor, MaxBndCoor)==1) {

                  outfile<<conc<<" "<<XLoc.at(i)<<" "<<YLoc.at(i)<<" "
                  <<ZLoc.at(i)<<endl;
                  outfile<<conc<<" "<<TempCord[3]<<" "
                              <<TempCord[4]<<" "
                              <<TempCord[5]<<endl;
                  conc++;

                  outfile7<<XLoc.at(i)<<" "<<YLoc.at(i)<<" "
                  <<ZLoc.at(i)<<endl;
                  outfile7<<TempCord[3]<<" "<<TempCord[4]<<" "
                  <<TempCord[5]<<endl;
                  outfile7<<-999<<" "<<-999<<" "<<-999<<endl;

                  //Include the Center Node in the Location and
Candidate Lists
                  XLoc.push_back(TempCord[3]);
                  YLoc.push_back(TempCord[4]);
                  ZLoc.push_back(TempCord[5]);

                  FlagL.push_back(1);

                  NodeConLoc.push_back(vector <double> (3));
                  NodeConLoc.back().at(0)=XLoc.back();
                  NodeConLoc.back().at(1)=YLoc.back();
                  NodeConLoc.back().at(2)=ZLoc.back();

                  CandNode.push_back(XLoc.size()-1);
                  NodeCon.at(i).push_back(XLoc.size()-1);

                  MotherN.push_back(i);

                  ThRes=SimThick(1,XLoc,YLoc,ZLoc,NodeCon,
                  MotherN.back(),thickness,VRange, MaxNumD, outfile12);

                  thickness.push_back(ThRes.at(0));
                  EstVar.push_back(ThRes.at(1));


                  outfile11<<nline<<" "<<XLoc.back()<<" "
                              <<YLoc.back()<<" "
                              <<ZLoc.back()<<" "
                              <<thickness.back()<<endl;
```

```
      TInter=ThickInt(NSeg,nline,XLoc.back(),YLoc.back(),ZLoc.back(),
                  thickness.back(),EstVar.back(),XLoc.at(i),YLoc.at(i),
            ZLoc.at(i),
                  thickness.at(i),EstVar.at(i),outfile11,outfile12);
                  outfile11<<nline<<" "<<XLoc.at(i)<<" "
                        <<YLoc.at(i)<<" "
                        <<ZLoc.at(i)<<" "<<thickness.at(i)<<endl;
                  nline=nline+1;

                  outfile8<<MotherN.back()<<" "<<FlagL.back()<<endl;

                  outfile4<<TempCord[3]<<" "
                        <<TempCord[4]<<" "
                        <<TempCord[5]<<endl;

                  outfile5<<TempCord[3]<<" "
                        <<TempCord[4]<<" "
                        <<TempCord[5]<<endl;

                  outfile10<<XLoc.back()<<" "<<YLoc.back()<<" "
                        <<ZLoc.back()<<" "<<thickness.back()<<endl;
            }
      }
} //End of for (i=0; i<n; i++)

//==========Calculation of Pair Relationships==================//
RetInd Config; //Observed Configuration

//Config.Ta=Indices of connected data locations
//Config.Tb=Indices of effective nodes on the template

vector<double> LagD;
vector<int> Indx_h;
vector<int> Indx_Azm;
vector<double> Temp_Azm; //Calculated Azimuth angle
vector<double> Temp_Dip; //Calculated Dip angle
vector<int> JIndx; //J indices
int LocSz=XLoc.size();

int ite=1;
int InSiz=XLoc.size(); //Size of the XLocs after initialization.
//========Main Loop Over the Data Locations=====//
vector <int> QDV;

while (CandNode.size()>=1 && ite<=inc_max) {

      if (ite<=InSiz-n) {
            i=0; //ite-1;
            SimNode.push_back(CandNode.at(i));

      CandNode.erase(CandNode.begin()+findI(CandNode.at(i),CandNode));
      }
      else {
```

```
      rnd=(CandNode.size())*rand_generator_(); //rand()
%CandNode.size();
      i=CandNode.at(rnd);
      SimNode.push_back(i);
      CandNode.erase(CandNode.begin()+findI(i,CandNode));
      }

//    cout<<"CandNodeSize="<<CandNode.size()<<endl;
      i=SimNode.back();
//    cout<<SimNode.size()<<" ";
      for (j=0; j<XLoc.size(); j++) {
            if (j!=i) {

      LagD.push_back(Cal_h(XLoc[i],YLoc[i],ZLoc[i],XLoc[j],YLoc[j],ZLoc
[j]));
      JIndx.push_back(j);
            Temp_Azm.push_back(Cal_Azm(XLoc[i],YLoc[i],ZLoc[i],
                                       XLoc[j],YLoc[j],ZLoc[j]));
            Temp_Dip.push_back(Cal_Dip(XLoc[i],YLoc[i],ZLoc[i],
                                       XLoc[j],YLoc[j],ZLoc[j]));
            }
      }

      vector< vector<double> > PatFreqs; //Store Frequencies of the
most similar patterns
      vector< vector<int> > SimPatF; //Store pattern number of the max.
similarity configuration
      vector< vector<int> > PatVals;// (NTemp,3); //Store the index and
freqeuncy of the max. similarity pattern and size of the observed
configuration
      vector <double> PatProbs; //Store pattern probabilities

      int sz=n;
      int it;
      double mysum;
      //Loop over Templates
      for (int jj=0; jj<NTemp; jj++) {

            //Store Current Template jj
            int abd=ReadTemp.at(jj).AzmAng.size(); //Template size

            vector<vector<double>>Template (abd,3);
            for (int ix=0; ix<ReadTemp.at(jj).AzmAng.size(); ix++) {

      Template[ix][0]=deg2rad(ReadTemp.at(jj).AzmAng.at(ix));

                  Template[ix][1]=find_temp_dip(XLoc.at(i),YLoc.at(i),Z
                  Loc.at(i),
                        DipMap); //Variable Template dip if dip map is
                  used
                  //Template[ix][1]=deg2rad(ReadTemp.at(jj).DipAng.at(i
            x));
                  //Template dip angle is used without a dip map
```

```
                  Template[ix][2]=ReadTemp.at(jj).Dist.at(ix);
            }

            //Scan with the Current Template jj; abd:Size of the
template
            Config=search_temp(Template,abd,LagD,Temp_Azm,Temp_Dip,
                              Dist_Tol,Azm_Tol,Dip_Tol);

//============================================================//
            //Current Template observes a Configuration successfully
            if (Config.Ta.empty()!=1)     {

                  //Store Observed Configuration
                  vector <int> ObsConfig (abd);
                  //cout<<"Config.Tb Nodes="<<" ";
                  for (int is=0; is<Config.Tb.size(); is++) {
                        cout<<Config.Tb.at(is)<<" ";
                        ObsConfig.at(Config.Tb.at(is))=1;
                  }
                  //cout<<endl;

                  //Calculate Similarity between Each Pattern in the
Histogram (jj) and the Observed Configuration
                  vector<int> Similarity; //Dot Product of ObsConfig
and PatConfig

                  for (int is=0; is<ReadHist.at(jj).PatNum.size();
is++) {
                        vector <int> PatConfig (abd);
                        for (int ix=0;
ix<ReadHist.at(jj).PatLen.at(is); ix++) {
      PatConfig.at(ReadPat.at(jj).Nodes.at(ReadHist.at(jj).PatNum.at(is
)).at(ix))=1;
                        }

      Similarity.push_back(DotProd(ObsConfig,PatConfig)); //Calculates
Similarity (dot product)
                        PatConfig.clear();

                  }

                  if(Similarity.empty()!=1) {

                        SimPatF.push_back(vector <int> ());
                        SimPatF.back()=findmax(Similarity); //Determine
the Patterns with the Max. Similarity

                        //Store frequencies of most similar patterns
with size greater than observed configuration
                        PatFreqs.push_back(vector <double>
(SimPatF.back().size()));

                        for (int is=0; is<SimPatF.back().size(); is++)
{
```

```
                        PatFreqs.back().at(is)=(ReadHist.at(jj).Freq.at
                        (SimPatF.back().at(is)));
                              PatVals.push_back(vector <int> (3));
                              PatVals.back().at(0)=jj; //Store Template
Number
                              PatVals.back().at(1)=
SimPatF.back().at(is); //Store Pattern Index
                              PatVals.back().at(2)=
                              ReadHist.at(jj).PatLen.at(SimPatF.back().
                        at(is));
                              //Store Pattern Length

                              PatProbs.push_back(double(sumall(ReadHist
                        .at(jj).Freq))); //Store Pattern Probability
                        }
                  }
            } //End of if(Config.Ta.empty()!=1)
      }//End of for (int jj=0; jj<NTemp; jj++)

      mysum=sumall(PatProbs);
      for (int is=0; is<PatProbs.size(); is++) {
            PatProbs.at(is)=PatProbs.at(is)/mysum;
      }

//================================================
      //Selection of Max. Probability Template and Calculation of New
Node Locations
      if (PatProbs.empty()!=1) {

                  vector <int> PatRes;
                  vector <int> MaxInd;

                  MaxInd=findmaxD(PatProbs);

            //    PatRes=SelectPat(PatVals, ReadHist,NTemp);
//PatRes[0]:Template Number, PatRes[1]:Pattern Number,
PatRes[2]:Pattern Frequency

            vector <int> SelPat=SelectPat(MaxInd,PatVals,
ReadHist,NTemp); //Select a Pattern from the selected
Histogram:SelPat(0)=Histogram Index;
            SelPat(1)=Selected Pattern Index
            int myind=SelPat.at(0); //Store Selected Histogram index
            cout<<"Selected Pattern Number="<<SelPat.at(1)<<" ";
            outfile6<<SelPat.at(1)<<endl;

            vector <int> mypatn; //Store Selected Pattern
            for (int is=0;
is<ReadHist.at(myind).PatLen.at(SelPat.at(1)); is++) {

      mypatn.push_back(ReadPat.at(myind).Nodes.at(ReadHist.at(myind).Pa
tNum.at(SelPat.at(1))).at(is));
            }
```

```
            int myi=myind;

            //Load Selected Template myind
            int abd=ReadTemp.at(myind).AzmAng.size(); //Template size

            vector<vector<double>>Template (abd,3);
            for (int ix=0; ix<ReadTemp.at(myind).AzmAng.size(); ix++) {

      Template[ix][0]=deg2rad(ReadTemp.at(myind).AzmAng.at(ix));

      Template[ix][1]=deg2rad(ReadTemp.at(myind).DipAng.at(ix));
                  Template[ix][2]=ReadTemp.at(myind).Dist.at(ix);

            }

            //Scan with the Selected Template myind; abd:Size of the
template
            Config=search_temp(Template,abd,LagD,Temp_Azm,Temp_Dip,
                              Dist_Tol,Azm_Tol,Dip_Tol);
            vector <int> TIndxTry;

            //*******Calculation of Pattern Node Locations and
      Expansion of Parameter Arrays***********//
                  double *NewNode;
                  vector<double> NodeTemp (3);

                  for (int is=0; is<ReadTemp.at(myind).AzmAng.size();
is++) {  //Loop over the template size
                              if (Config.Ta.empty()!=1)      {
                                    if (findE(is,mypatn)==1) {

                                          if(findE(is,Config.Tb)==0) {

      Dipp=find_temp_dip(XLoc.at(i),YLoc.at(i),ZLoc.at(i),DipMap);
                  //ReadTemp.at(myind).DipAng.at(is) is replaced by
Dipp

      NewNode=BackCal2(ReadTemp.at(myind).AzmAng.at(is),Dipp,
                        ReadTemp.at(myind).Dist.at(is),XLoc[i],YLoc[i],
                  ZLoc[i],
                        Dist_Tol,Azm_Tol,Dip_Tol);

      NodeTemp.at(0)=NewNode[0];
      NodeTemp.at(1)=NewNode[1];
      NodeTemp.at(2)=NewNode[2];

      AzmAngVals.push_back(NewNode[3]);
                  if (BndContl(NodeTemp,MinBndCoor,MaxBndCoor)==1){
                        if (findED(NewNode[0],XLoc)==0 &&
             findED(NewNode[1],YLoc)==0 && findED(NewNode[2],ZLoc)==0) {

      XLoc.push_back(NewNode[0]);
      YLoc.push_back(NewNode[1]);
      ZLoc.push_back(NewNode[2]);
```

```
      outfile4<<NewNode[0]<<" "
            <<NewNode[1]<<" "
            <<NewNode[2]<<endl;
      MotherN.push_back(MotherN.at(i-n));
      NodeCon.at(MotherN.at(i-n)).push_back(XLoc.size()-1);

      outfile<<conc<<" "<<XLoc.at(i)<<" "<<YLoc.at(i)<<" "
                        <<ZLoc.at(i)<<endl;
      outfile<<conc<<" "<<NewNode[0]<<" "<<NewNode[1]<<" "
                                    <<NewNode[2]<<endl;

      outfile7<<XLoc.at(i)<<" "<<YLoc.at(i)<<" "<<ZLoc.at(i)<<endl;
      outfile7<<NewNode[0]<<" "<<NewNode[1]<<" "<<NewNode[2]<<endl;
      outfile7<<-999<<" "<<-999<<" "<<-999<<endl;

      outfile5<<NewNode[0]<<" "<<NewNode[1]<<" "<<NewNode[2]<<endl;
      conc++;


      CandNode.push_back(XLoc.size()-1); //Expand CandNode Vector and
add the new Location
      FlagL.push_back(1);

      NodeConLoc.push_back(vector <double> (3));
      NodeConLoc.back().at(0)=XLoc.back();
      NodeConLoc.back().at(1)=YLoc.back();
      NodeConLoc.back().at(2)=ZLoc.back();

      outfile8<<MotherN.back()<<" "<<FlagL.back()<<endl;

      ThRes=SimThick(1,XLoc,YLoc,ZLoc, NodeCon,MotherN.back(),
                        thickness,VRange, MaxNumD, outfile12);

      thickness.push_back(ThRes.at(0));
      EstVar.push_back(ThRes.at(1));

      outfile11<<nline<<" "<<XLoc.back()<<" "<<YLoc.back()<<" "
            <<ZLoc.back()<<" "<<thickness.back()<<endl;


      TInter=ThickInt(NSeg,nline,XLoc.back(),YLoc.back(),ZLoc.back(),th
ickness.back(),EstVar.back(),XLoc.at(i),YLoc.at(i),ZLoc.at(i),thickness
.at(i),EstVar.at(i),outfile11,outfile12);
      outfile11<<nline<<" "<<XLoc.at(i)<<" "<<YLoc.at(i)<<" "
                        <<ZLoc.at(i)<<" "
                        <<thickness.at(i)<<endl;
      nline=nline+1;
      outfile10<<XLoc.back()<<" "<<YLoc.back()<<" "
                  <<ZLoc.back()<<" "<<thickness.back()<<endl;

                        } //End of if findED(NewNode[0],XLoc)==0
                  } //End of if
(BndContl(NodeTemp,MinBndCoor,MaxBndCoor)==1)
```

```
            } //End of if(findE(is,Config.Tb)==0)
            else {

            int Tind=findI(is,Config.Tb);

            if (FlagL.at(SimNode.back())==1 ||
                  FlagL.at(JIndx.at(Config.Ta.at(Tind)))==1) {

            if
(findE(JIndx.at(Config.Ta.at(Tind)),NodeCon.at(MotherN.at(i-n)))==0
                  &&  MotherN.at(i-n)!=JIndx.at(Config.Ta.at(Tind))) {

            outfile<<conc<<" "
                  <<XLoc.at(SimNode.back())<<" "
                  <<YLoc.at(SimNode.back())<<" "
                  <<ZLoc.at(SimNode.back())<<endl;


      outfile7<<XLoc.at(SimNode.back())<<" "
            <<YLoc.at(SimNode.back())<<" "
            <<ZLoc.at(SimNode.back())<<endl;

      outfile<<conc<<" "
            <<XLoc.at(JIndx.at(Config.Ta.at(Tind)))<<" "
            <<YLoc.at(JIndx.at(Config.Ta.at(Tind)))<<" "
            <<ZLoc.at(JIndx.at(Config.Ta.at(Tind)))<<endl;


      outfile7<<XLoc.at(JIndx.at(Config.Ta.at(Tind)))<<" "
            <<YLoc.at(JIndx.at(Config.Ta.at(Tind)))<<" "
            <<ZLoc.at(JIndx.at(Config.Ta.at(Tind)))<<endl;

      outfile7<<-999<<" "<<-999<<" "<<-999<<endl;

      outfile5<<XLoc.at(JIndx.at(Config.Ta.at(Tind)))<<" "
            <<YLoc.at(JIndx.at(Config.Ta.at(Tind)))<<" "
            <<ZLoc.at(JIndx.at(Config.Ta.at(Tind)))<<endl;


      outfile10<<XLoc.at(JIndx.at(Config.Ta.at(Tind)))<<" "
            <<YLoc.at(JIndx.at(Config.Ta.at(Tind)))<<" "
            <<ZLoc.at(JIndx.at(Config.Ta.at(Tind)))<<" "
            <<thickness.at(JIndx.at(Config.Ta.at(Tind)))<<endl;

      if (FlagL.at(SimNode.back())==1) {

            if (JIndx.at(Config.Ta.at(Tind))>=n) {
                  if (findE(JIndx.at(Config.Ta.at(Tind)),
                        NodeCon.at(MotherN.at(i-n)))==0)    {

      NodeCon.at(MotherN.at(i-
n)).push_back(JIndx.at(Config.Ta.at(Tind))); }
            }
```

```
            else {

if (findE(SimNode.back(),NodeCon.at(JIndx.at(Config.Ta.at(Tind))))==0)
      {

      NodeCon.at(JIndx.at(Config.Ta.at(Tind))).push_back(SimNode.back()
);}
            }
      }

            if (FlagL.at(SimNode.back())==0) {

                  if (JIndx.at(Config.Ta.at(Tind))>=n) {

      if
(findE(SimNode.back(),NodeCon.at(MotherN.at(JIndx.at(Config.Ta.at(Tind)
-n))))==0) {

NodeCon.at(MotherN.at(JIndx.at(Config.Ta.at(Tind)-
n))).push_back(SimNode.back()); }
                  }

                  else {
                  if
(findE(SimNode.back(),NodeCon.at(MotherN.at(JIndx.at(Config.Ta.at(Tind)
))))==0){
      NodeCon.at(JIndx.at(Config.Ta.at(Tind))).push_back(SimNode.back()
);}
                  }
            }

      conc++;

            }
      }
      }  //End of if(findE(is,Config.Tb)==0) else
}//End of if (findE(is,mypatn)==1)
            else {
                  if(findE(is,Config.Tb)==0) {
      //ReadTemp.at(myind).DipAng.at(is) is replaced by Dipp for Dip
map implementation.
      Dipp=find_temp_dip(XLoc.at(i),YLoc.at(i),ZLoc.at(i),DipMap); //If
not using dip map, use the following line and comment this line.

      //Dipp=ReadTemp.at(myind).DipAng.at(is);

      NewNode=BackCal2(ReadTemp.at(myind).AzmAng.at(is),Dipp,ReadTemp.a
t(myind).Dist.at(is),XLoc[i],YLoc[i],ZLoc[i],Dist_Tol,Azm_Tol,Dip_Tol);


      NodeTemp.at(0)=NewNode[0];
      NodeTemp.at(1)=NewNode[1];
      NodeTemp.at(2)=NewNode[2];
```

```
            if (BndContl(NodeTemp,MinBndCoor,MaxBndCoor)==1){

         if (findED(NewNode[0],XLoc)==0 && findED(NewNode[1],YLoc)==0 &&
                                           findED(NewNode[2],ZLoc)==0) {

      XLoc.push_back(NewNode[0]);
      YLoc.push_back(NewNode[1]);
      ZLoc.push_back(NewNode[2]);

      CandNode.push_back(XLoc.size()-1); //Expand CandNode Vector and
add the new Location
      MotherN.push_back(MotherN.at(i-n));
      FlagL.push_back(0);
      thickness.push_back(0);
      EstVar.push_back(1);

      outfile8<<MotherN.back()<<" "<<FlagL.back()<<endl;

      outfile4<<NewNode[0]<<" "
            <<NewNode[1]<<" "
            <<NewNode[2]<<endl;
                                                      }
                                                   }
                                             }
                                             else {
                                                   int
Tind=findI(is,Config.Tb);

                                       if (FlagL.at(SimNode.back())==1 ||
                              FlagL.at(JIndx.at(Config.Ta.at(Tind)))==1) {


      if(findE(JIndx.at(Config.Ta.at(Tind)),
            NodeCon.at(MotherN.at(i-n)))==0 &&
            MotherN.at(i-n)!=JIndx.at(Config.Ta.at(Tind))) {

      outfile<<conc<<" "
            <<XLoc.at(SimNode.back())<<" "
            <<YLoc.at(SimNode.back())<<" "
            <<ZLoc.at(SimNode.back())<<endl;

      outfile7<<XLoc.at(SimNode.back())<<" "
            <<YLoc.at(SimNode.back())<<" "
            <<ZLoc.at(SimNode.back())<<endl;

      outfile<<conc<<" "
            <<XLoc.at(JIndx.at(Config.Ta.at(Tind)))<<" "
            <<YLoc.at(JIndx.at(Config.Ta.at(Tind)))<<" "
            <<ZLoc.at(JIndx.at(Config.Ta.at(Tind)))<<endl;

      outfile7<<XLoc.at(JIndx.at(Config.Ta.at(Tind)))<<" "
            <<YLoc.at(JIndx.at(Config.Ta.at(Tind)))<<" "
            <<ZLoc.at(JIndx.at(Config.Ta.at(Tind)))<<endl;
```

```
      outfile7<<-999<<" "<<-999<<" "<<-999<<endl;

      outfile5<<XLoc.at(JIndx.at(Config.Ta.at(Tind)))<<" "
            <<YLoc.at(JIndx.at(Config.Ta.at(Tind)))<<" "
            <<ZLoc.at(JIndx.at(Config.Ta.at(Tind)))<<endl;

      outfile10<<XLoc.at(JIndx.at(Config.Ta.at(Tind)))<<" "
            <<YLoc.at(JIndx.at(Config.Ta.at(Tind)))<<" "
            <<ZLoc.at(JIndx.at(Config.Ta.at(Tind)))<<" "
            <<thickness.at(JIndx.at(Config.Ta.at(Tind)))<<endl;
      if (thickness.at(i)!=0) {

      outfile11<<nline<<" "<<XLoc.at(i)<<" "<<YLoc.at(i)<<" "
            <<ZLoc.at(i)<<" "
            <<thickness.at(i)<<endl;


      TInter=ThickInt(NSeg,nline,XLoc.at(JIndx.at(Config.Ta.at(Tind))),
YLoc.at(JIndx.at(Config.Ta.at(Tind))),ZLoc.at(JIndx.at(Config.Ta.at(Tin
d))),thickness.back(),EstVar.back(),XLoc.at(i),YLoc.at(i),ZLoc.at(i),th
ickness.at(i),EstVar.at(i),outfile11,outfile12);
                                                                  }
      nline=nline+1;
      if (FlagL.at(SimNode.back())==1) {
            if (JIndx.at(Config.Ta.at(Tind))>=n) {
                  if (findE(JIndx.at(Config.Ta.at(Tind)),
                    NodeCon.at(MotherN.at(i-n)))==0){

      NodeCon.at(MotherN.at(i-
n)).push_back(JIndx.at(Config.Ta.at(Tind))); }
            }
            else {

      if
(findE(SimNode.back(),NodeCon.at(JIndx.at(Config.Ta.at(Tind))))==0)
      {

      NodeCon.at(JIndx.at(Config.Ta.at(Tind))).push_back(SimNode.back()
); }
            }

      }

      if (FlagL.at(SimNode.back())==0) {
            if (JIndx.at(Config.Ta.at(Tind))>=n) {

            if (findE(SimNode.back(),
            NodeCon.at(MotherN.at(JIndx.at(Config.Ta.at(Tind)-n))))==0
            ){
           NodeCon.at(MotherN.at(JIndx.at(Config.Ta.at(Tind)-
                  n))).push_back(SimNode.back()); }

      }
      else {
```

```
      if
(findE(SimNode.back(),NodeCon.at(MotherN.at(JIndx.at(Config.Ta.at(Tind)
))))==0){

      NodeCon.at(JIndx.at(Config.Ta.at(Tind))).push_back(SimNode.back()
);}
            }
      }
conc++;
      }

      }
      }
}//End of if (findE(is,mypatn)==1) else
} //End of if(Config.Ta.empty()!=1)
//End of for (int is=0; is<ReadTemp.at(myind).AzmAng.size(); is++)

} //End of if(PatProbs.empty()!=1)

      //If PatProbs.empty();
      else {
            int myind=0;
            double *NewNode;
            vector<double> NodeTemp (3);
            int abd=ReadTemp.at(myind).AzmAng.size(); //Template size
            vector<vector<double>>Template (abd,3);

            for (int ix=0; ix<ReadTemp.at(myind).AzmAng.size(); ix++) {

      Template[ix][0]=deg2rad(ReadTemp.at(myind).AzmAng.at(ix));

      Template[ix][1]=deg2rad(ReadTemp.at(myind).DipAng.at(ix));
                  Template[ix][2]=ReadTemp.at(myind).Dist.at(ix); }

            for (int is=0; is<ReadTemp.at(myind).AzmAng.size(); is++) {
//Loop over the template size

                  //ReadTemp.at(myind).DipAng.at(is) is replaced by
Dipp

      Dipp=find_temp_dip(XLoc.at(i),YLoc.at(i),ZLoc.at(i),DipMap);//If
not using dip map, use the following line and comment this line.
      //Dipp=ReadTemp.at(myind).DipAng.at(is);

      NewNode=BackCal2(ReadTemp.at(myind).AzmAng.at(is),Dipp,ReadTemp.a
t(myind).Dist.at(is),XLoc[i],YLoc[i],ZLoc[i],Dist_Tol,Azm_Tol,Dip_Tol);
                  NodeTemp.at(0)=NewNode[0];
                  NodeTemp.at(1)=NewNode[1];
                  NodeTemp.at(2)=NewNode[2];

                  if (BndContl(NodeTemp,MinBndCoor,MaxBndCoor)==1){

                        if (findED(NewNode[0],XLoc)==0 &&
```

```
findED(NewNode[1],YLoc)==0 && findED(NewNode[2],ZLoc)==0) {

                        XLoc.push_back(NewNode[0]);
                        YLoc.push_back(NewNode[1]);
                        ZLoc.push_back(NewNode[2]);

                        CandNode.push_back(XLoc.size()-1); //Expand
CandNode Vector and add the new Location
                        FlagL.push_back(0);

                        thickness.push_back(0);
                        EstVar.push_back(1);

                        MotherN.push_back(MotherN.at(i-n));
                        outfile8<<MotherN.back()<<"
"<<FlagL.back()<<endl;

                        outfile4<<NewNode[0]<<" "<<NewNode[1]<<" "
                              <<NewNode[2]<<endl;
                        }
                  }
            }

      }//End of else if(PatProbs.empty()!=1)

            LagD.clear();
            JIndx.clear();
            Temp_Azm.clear();
            Temp_Dip.clear();
            Config.Ta.clear();
            Config.Tb.clear();

            cout<<"CandNodeSize="<<CandNode.size()<<"Iteration="<<" "
                  <<ite<<endl;

            //Calculate Euclidean Distance Between the Histograms at
Every ScanStep
            if (ite%ScanStep==0 || ite==inc_max-1) {
            for (int hi=0; hi<ReadHist.size(); hi++) {
                  int abd=ReadTemp.at(hi).AzmAng.size(); //Template
size
            vector<vector<double>>Template (abd,3);
            for (int ix=0; ix<ReadTemp.at(hi).AzmAng.size(); ix++) {

      Template[ix][0]=deg2rad(ReadTemp.at(hi).AzmAng.at(ix));
      Template[ix][1]=deg2rad(ReadTemp.at(hi).DipAng.at(ix));
      Template[ix][2]=ReadTemp.at(hi).Dist.at(ix);
      }

      MyDist.at(hi).push_back(Distance(XLoc,YLoc, ZLoc,
            Template,FlagL,ReadHist,hi,Dist_Tol,Azm_Tol,Dip_Tol));

//===This section is used when histogram deviation stopping criteria is
used===
```

```
//if (MyDist.at(hi).size()!=1) {
      // if
(MyDist.at(hi).back()>MyDist.at(hi).at(MyDist.at(hi).size()-2) ||
      MyDist.at(hi).back()==MyDist.at(hi).at(MyDist.at(hi).size()-2)) {
                        //if (fabs(MyDist.at(hi).back()-
MyDist.at(hi).at(MyDist.at(hi).size()-
2))/MyDist.at(hi).at(MyDist.at(hi).size()-2)<0.0025) {
            //ite=inc_max;
                        //break;}
                        //}
            //}
      }
}

            //===========Quadrant Check============
      if (ite%checkstep==0) {

      CandNode=QAnalysis(XLoc,YLoc,ZLoc,Quad_Num.at(0),Quad_Num.at(1),
                  Quad_Num.at(2),Orig_Quadrant,SimNode,QuadLim); }
      ite=ite+1;
}

cout<<"SimNodeSize="<<" "<<SimNode.size()<<endl;

//Store the node connections
for (int ix=0; ix<n; ix++) {
      outfile2<<ix<<" ";
      if (NodeCon.at(ix).empty()!=1) {
            for (int is=0; is<NodeCon.at(ix).size(); is++) {
                  outfile2<<NodeCon.at(ix).at(is)<<" ";
            }
      }
      outfile2<<endl;
}

//Store Azimuth Angle Values for any additional analysis.
for (int ix=0; ix<AzmAngVals.size(); ix++) {
      outfile9<<AzmAngVals.at(ix)<<endl;
}

////================================================//

cout<<"Finished!"<<endl;

return 0;
} //End of Main
```

References

Afifi, A.M., 2005, Ghawar: The Anatomy of the World's Largest Oil Field, AAPG Distinguished Lecture, Search and Discovery Article 20026.

Al Shahri, A.M., Al Muraikhi, A., 1998, SPE 49275, A Novel Approach to Characterize Dynamic Interaction Between Super Permeability Layers with Vertical Faults and Their Effect on Flood Front Movement, Proceeding of SPE ATCE, New Orleans, Louisiana, 27-30 September.

Alabert, F.G., Aquitaine, E., Modot, V., 1992, SPE 24893, Stochastic Models of Reservoir Heterogeneity: Impact on Connectivity and Average Permabilities, Proceeding of 67th SPE Annual Technical Conference and Exhibition, Washington D.C., 4-7 October.

Allmendinger, R.W., 2011, Stereonet 7 for Windows Beta.

Arpat, G.B., Caers, J., 2007, Conditional Simulation with Patterns, Mathematical Geology, Vol. 39, No.2.

Behnken, F.,H., White, J., 2007, A Different Production Paradigm Brings a Giant Back to Life: Yates Field, Pecos County, TX, Proceeding of 2007 West Texas Geological Society Fall Symposium, 24-26 October, Midland, TX.

Biver, P., Pivot, F., Henrion, V., 2012, Estimation of Most Likely Lithology Map in the Context of Truncated Gaussian Techniques, Proceeding of 9th Geostatistics Congress, Oslo, Norway, 11-15 June.

Borghi, A., Renard, P., Jenni, S., 2012, A Pseudo-Genetic Stochastic Model to Generate Karstic Networks, Journal of Hydrology, Vol. 414-415, p.516-529.

Botton-Dumay R, Manivit T, Massonnat G, Gay V, 2002, SPE 78534, Karstic High Permeability Layers: Characterization and Preservation while Modeling Carbonate Reservoirs, Proceeding of 10th IPEC Meeting, Abu Dhabi, UAE.

Caceres A., Emery, X., Riquelme, R., 2010, Truncated Gaussian Kriging as an Alternative to Indicator Kriging, Proceeding of 4th International Conference on Mining Innovation, Santiago, Chile, 23-25 June.

Caers, J., 2002, SPE 77429, Geostatistical History Matching Under Training-Image Based Geological Modeling Constraints, Proceeding of SPE ATCE, San Antonio, Texas, 29 September- 2 October.

Caers, J., Zhang, T., 2004, Multiple-Point Geostatistics: A Quantitative Vehicle for Integrating Geologic Analogs into Multiple Reservoir Models, *in*, Grammer, G.M., Harris, P.M., Eberli, G.P., eds., Integration of Outcrop and Modern Analogs in Reservoir Modeling, AAPG Memoir, 80, p.383-394.

Caers, J.S., Srinivasan, S., Journel, A.G., 2000, SPE 66310, Geostatistical Quantification of Geological Information for a Fluvial Type North Sea Reservoir, SPE Reservoir Evaluation and Engineering, V.3, p.457-467.

Campanelle, J.D., Wadleigh, E.E., Gilman, J.R., 2000, SPE 58996, Flow Characterization- Critical for Efficiency of Field Operations and IOR, Proceeding of SPE International Petroleum Conference and Exhibition, 1-3 February, Villahermosa, Mexico.

Cantrell, D., Swart, P., Hagerty, R., 2004, Genesis and Characterization of Dolomite, Arab-D Reservoir, Ghawar Field, Saudi Arabia, GeoArabia, Vol.9, No.2, p.11-36.

Cheng, A.M., Kwan, J.T., 2012, SPE 154270, Optimal Injection Design Utilizing Tracer and Simulation in a Surfactant Pilot for a Fractured Carbonate Yates Field, Proceeding of SPE Improved Oil Recovery Symposium, 14-18 April, Tulsa, OK.

Choquette, P.W., James, N.P., 1988, Paleokarst, Springer – Verlag New York Inc., p. 1-21.

CMG-IMEX, 2011, User's Guide, IMEX Implicit-Explicit Black Oil Simulator, Computer Modeling Group Ltd., Calgary, Canada.

Craig, D.H., 1988, Caves and Other Features of Permian Karst in San Andres Dolomite, Yates Field Reservoir, West Texas, *in* James, N.P., Choquette, P.W., eds., Paleokarst, Springer – Verlag New York Inc., p. 342-363.

Craig, D.H., Mruk, D.H., Heymans, M.J., Crevello, P.D., Lanz, R.C., 1986, Stratigraphy and Reservoir Geology of the San Andres Dolomite – Yates Field, West Texas, *in*, Bebout, D.G., Harris, P.M., eds., Hydrocarbon Reservoir Studies San Andres/Grayburg Formations, Permian Basin, SEPM Publication, 86-26, p.139-143.

Da Veiga, S., Le Ravalec, M., 2010, SPE 130976, Rebuilding Existing Geological Models, Proceeding of SPE EUROPEC/EAGE Annual Conference and Exhibition, Barcelona, Spain, 14-17 June.

Daniel, E.J., 1954, Fractured Reservoirs of Middle East, AAPG Bulletin, V.38, No.5, p. 774-815.

Davies, G.R., Smith Jr., L.B., 2006, Structurally Controlled Hydrothermal Dolomite Reservoir Facies: An Overview, AAPG Bulletin, V.90, No.11, p.1641-1690.

Dembicki, E.A., Machel, H.G., 1996, Recognition and Delineation of Paleokarst Zones by the Use of Wireline Logs in the Bitumen-Saturated Upper Devonian Grosmont Formation of Northeastern Alberta, Canada, AAPG Bulletin, 80, No:5, p.695-712.

Deutsch, C.V., Journel, A., 1998, GSLIB-Geostatistical Software Library and User's Guide, 2nd Edition, Oxford University Press, New York.

Dogru, A.H., Dreiman, W.T., Hemanthkumar, K., Fung, L.S., 2001, Simulation of Super-K Behavior in Ghawar by a Multi-Million Cell Parallel Simulator, SPE 68066, Proceeding of SPE Middle East Oil & Gas Show and Conference, Kingdom of Bahrain, 17-20 March.

Dreybrodt, W., Gabrovsek, F., 2002, Basic Processes and Mechanisms Governing the Evolution of Karst, *in* Gabrovsek, F., eds., Evolution of Karst: From Prekarst to Cessation, 2002, Proceedings of the Symposium Evolution of Karst: From Prekarst to Cessation, Postojna, Slovenia, September 17-21.

Eskandaridalvand, K., 2008, Growthsim: A Complete Framework for Integrating Static and Dynamic Data into Reservoir Models, PhD. Dissertation, The University of Texas at Austin, 220 pages.

Esteban, M., Wilson, J.L., 1993, Introduction to Karst Systems and Paleokarst Reservoirs, *in* Fritz, D., Wilson, J.L., Yurewicz, D.A., eds., Paleokarst Related Hydrocarbon Reservoirs, SEPM Core Workshop, No.18, New Orleans, April 25.

Erzeybek, S., Srinivasan, S., 2011, Modeling Connected Geologic Features Using Multiple-Point Statistics from Non-Gridded Data, Proceeding of Annual Conference of the International Association for Mathematical Geosciences, Salzburg, Austria, 5-9 September.

Erzeybek, S., Srinivasan, S., Janson, X., 2012, Multiple-Point Statistics in a Non-Gridded Domain: Application to Karst/Fracture Modeling, *in* Abrahamsen, P., Hauge, R., Kolbjørnsen, O., eds., Geostatistics Oslo 2012, Springer, New York.

Filipponi, M., Jeannin, P.Y., Tacher, L., 2009, Evidence of Inception Horizons in Karst Conduit Networks, Geomorphology, Volume 106, p.86-99.

Flodin, E.A., Aydin, A., 2004, Evolution of a Strike-Slip Fault Network, Valley of Fire State Park, Southern Nevada, GSA Bulletin, 116, No.1-2, p.42-59.

Ford, D., 1988, Characteristics of Dissolutional Cave Systems in Carbonate Rocks, *in* James, N.P., Choquette, P.W., eds., Paleokarst, Springer – Verlag New York Inc., p. 25-57.

Ford, D.C., 1989, Features of the Genesis of Jewel Cave and Wind Cave, Black Hills, South Dakota, NSS Bulletin, 51, p.100-110.

Gilman, J.R., Bowzer, J.L., Rothkopf, B.W., 1995, SPE 28568, Application of Short-Radius Horizontal Boreholes in the Naturally Fractured Yates Field, SPE Reservoir Engineering Journal, Volume 10, No.1, p.10-15.

Goovaerts, P., 1997, Geostatistics for Natural Resources Evaluation: Oxford University Press.

Goovaerts, P., 1998, Ordinary Cokriging Revisited, Mathematical Geology, Volume 30, No.1, p.21-42.

GsTL: Geostatistical Template Library, Center for Reservoir Forecasting, Stanford University.

Henrion V, Pellerin J, Caumon G, 2008, A Stochastic Methodology for 3D Cave System Modeling, Proceeding of 8th International Geostatistics Congress, Santiago, Chile, 1-5 December.

Hill, M.E., Martin, A., 2008, A Conceptual Model of Preferential Flow Pathways in a Karst Aquifer Using Multiple Data Types, *in* Sasowsky, I.R., Feazel, C.T., Mylroie, J.E., Palmer, A.N., Palmer, M.V., eds., Karst From Recent to Reservoirs, Karst Waters Institute Special Publication, No.14, p.95-103.

Horrocks, R.D., Szukalski, B.W., 2002, Using Geographic Information Systems to Develop a Cave Potential Map for Wind Cave, South Dakota, Journal of Cave and Karst Studies, V.64, No:1, p.63-70.

Kanungo, T., Mount, D.M., Netanyahu, N.S., Piatko, C.D., Silverman, R., Wu., A.Y., 2002, An Efficient k-Means Clustering Algorithm: Analysis and Implementation, IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 24, No.7, p.881-892.

Kerans, C., 1988, Karst-Controlled Reservoir Heterogeneity in Ellenburger Group Carbonates of West Texas, AAPG Bulletin, Volume 72, No.10, p.1160-1183.

Labourdette, R., Lascu, I., Mylroie, J., Roth, M., 2007, Process Like Modeling of Flank-Margin Caves: From Genesis to Burial Evolution, Journal of Sedimentary Research, V.77, p.965- 979.

Lanzarini, W.L., Poletto, C.A., Tavares, G., Pesco, S., Lopes, H., 1997, SPE 38953, Stochastic Modeling of Geometric Objects and Reservoir Heterogeneities, Proceeding of 5th Latin American and Caribbean Petroleum Engineering Conference and Exhibition, 30 August-3 September, Rio De Janeiro, Brazil.

Le Ravalec-Dupin, M., Roggero, F., Froidevaux, R., 2004, Conditioning Truncated Gaussian Realizations to Static and Dynamic Data, SPE Journal, December 2004.

Levine, S., Sigmon, R., Douglas, S., 2002, Yates Field-Super Giant of the Permian Basin, Houston Geological Society Bulletin, November, p.39-51.

Li, J., Zhang, W., Luo, X., Hu, G., 2008, Paleokarst Reservoirs and Gas Accumulation in the Jiangbian Field, Ordos Basin, Marine and Petroleum Geology, 25, p.401-415.

Liu, X., Srinivasan, S., 2005, Field Scale Stochastic Modeling of Fracture Networks, *in* Leuangthong, O., Deutsch, C.V., eds., Geostatistics Banff 2004, Springer, p.75-84.

Liu, X., Zhang, C., Liu, Q., Birkholzer, J., 2009, Multiple-point Statistical Prediction on Fracture Networks at Yucca Mountain, Environmental Geology, V.57, p.1361-1370.

Loucks, R.G., 1999, Paleocave Carbonate Reservoirs: Origins, Burial-Depth Modifications, Spatial Complexity and Reservoir Implications, AAPG Bulletin, 83, No: 11, 1795-1834.

Loucks, R.G., 2007, A Review of Coalesced, Collapsed-Paleocave Systems and Associated Suprastratal Deformation, Time in Karst, Postojna, p.121-132.

Loucks, R.G., Handford, C.R., 1992, Origin and Recognition of Fractures, Breccias and Sediment Fills in Paleocave Reservoir Networks, *in* Candelaria, M.P., Reed, C.L., eds., Paleokarst, Karst Related Diagenesis and Reservoir Development: Examples from Ordovician-Devonian Age Strata of West Texas and the Mid-Continent, SEPM Permian Basin Section, 1992, Annual Field Trip Franklin Mountains., El-Paso, Texas, April 9-11, p.31-44.

Loucks, R.G., Mescher, P.K., 2001, Paleocave Facies Classification and Associated Pore Types, Proceeding of AAPG Southwest Section, Annual Meeting, Dallas, Texas, March 11-13.

Massonat, G., Pernarcic, E., 2002, SPE 77591, Assessment and Modeling of High Permeability Areas in Carbonate Reservoirs, Proceeding of SPE ATCE, 29 September – 2 October, San Antonio, Texas.

Meyers, W.J., 1988, Paleokarstic Features on Mississippian Limestones, New Mexico, *in* James, N.P., Choquette, P.W., eds., Paleokarst, Springer – Verlag New York Inc., p. 306-328.

Miller, T.E., 1989, Evidence of Quaternary Tectonic Activity and for Regional Aquifer Flow at Wind Cave, South Dakota, NSS Bulletin, V.52, p.111-119.

Palmer, A.N., 1991, Origin and Morphology of Limestone Caves, Geological Society of America Bulletin, 103, 1-21.

Palmer, A.N., Palmer, M.V., 2008, Field Guide to the Paleokarst of the Southern Black Hills, *in* Sasowsky, I.R., Feazel, C.T., Mylroie, J.E., Palmer,A.N., Palmer, M.V., eds., Karst From Recent to Reservoirs, Karst Waters Institute Special Publication, No.14, p.189-220.

Pyrcz, M., Deutsch, C.V., 2001, Two Artifacts of Probability Field Simulation, Mathematical Geology, Vol. 33, No.7.

Remy, N.N., Wu, J., Boucher, A., 2008, Applied Geostatistics with SGeMS: A User's Guide, Cambridge, UK, Cambridge University Press.

Smart, P.L., Palmer, R.J., Whitaker, F., Wright, V.P., 1988, Neptunian Dikes and Fissure Fills: An Overview and Account of Some Modern Examples, *in* James, N.P., Choquette, P.W., eds., Paleokarst, Springer – Verlag New York Inc., p. 150-151.

Spencer, A.W., Warren, J.K., 1986, Depositional Styles in the Queen and Seven Rivers Formations – Yates Field, Pecos Co, Texas, *in* Bebout, D.G., Harris, P.M., eds., Hydrocarbon Reservoir Studies San Andres/Grayburg Formations, Permian Basin, SEPM Publication, 86-26, p.135-137.

Srivastava, R.M., 1992, SPE 24573, Reservoir Characterization with Probability Field Simulation, Proceeding of 67th SPE Annual Technical Conference and Exhibition, Washington D.C., 4-7 October.

Stafford, K.W., Behnken, F.H., White, J.G., Hypogene Speleogenesis within the Central Basin Platform: Karst Porosity in the Yates Field, Pecos County, Texas, USA, *in* Sasowsky, I.R., Feazel, C.T., Mylroie, J.E., Palmer,A.N., Palmer, M.V., eds., Karst From Recent to Reservoirs, Karst Waters Institute Special Publication, No.14, p.174-178.

Strebelle, S., 2002, Conditional Simulation of Complex Geological Structures Using Multiple-Point Statistics, Mathematical Geology, V.34, No.1.

Tinker, S.W., Ehrets, J.R., Brondos, M.D., 1995, Multiple Karst Events Related to Stratigraphic Cyclicity: San Andres Formation, Yates Field, West Texas, *in* Budd, D.A., Saller, A.H., Harris, P.M., eds., Unconformities and Porosity in Carbonate Strata, AAPG Memoir, 63, p.213 – 237.

Tinker, S.W., Mruk, D.,H., 1995, Reservoir Characterization of a Permian Giant: Yates Field, West Texas, *in* Stoudt, E.L., Harris, P.M., eds., Hydrocarbon Reservoir Characterization: Geologic Framework and Flow Unit Modeling, SEPM Short Course, No.34, 4-5 March, Houston, TX.

Trice, R., 2005, SPE 93679, Challenges and Insights in Optimizing Oil Production from Middle East Mega Karst Reservoirs, Proceeding of 14th SPE Middle East Oil & Gas Show and Conference, Bahrain, 12-15 March.

Uba, H.M., Chiffoleau, Y., Pham, T., Divry, V., Kaabi, A., Thuwaini, J., 2007, SPE 105560, Application of a Hybrid Dual-Porosity/ Dual-Permeability Representation of Large-Scale Fractures to the Simulation of a Giant Carbonate Reservoir, Proceeding of 15th SPE Middle East Oil & Gas Show and Conference, Kingdom of Bahrain, 11-14 March

Wright, V.P., 2002, Dissolution and Porosity Development in Carbonates, *in* Gabrovsek, F., eds., Evolution of Karst: From Prekarst to Cessation, 2002, Proceedings of the Symposium Evolution of Karst: From Prekarst to Cessation, Postojna, Slovenia, September 17-21.

Wu, J., Boucher, A., Zhang, T., 2008, A SGeMS Code for Pattern Simulation of Continuous and Categorical Variables: FILTERSIM, Computers and Geosciences, V.34, p.1863-1876.

Xu, W., Journel, A.G., 1993, SPE 27412, GTSIM: Gaussian Truncated Simulations of Reservoir Units in a West Texas Carbonate Field.

Vita

Selin Erzeybek Balan was born in Turkey. She earned her B.S. and M.S. degrees in Petroleum and Natural Gas Engineering, and her minor degree in Earth Sciences from Middle East Technical University (METU) in Ankara, Turkey. Upon completion of her M.S. education, she enrolled in Ph.D. program in Petroleum and Geosystems Engineering at the University of Texas at Austin.

Email: selinerzeybek@gmail.com

This dissertation was typed by Selin Erzeybek Balan.